DATA STORAGE HIERARCHY SYSTEMS FOR DATA BASE COMPUTERS

by

Chat-Yu Lam

B.Sc. Massachusetts Institute of Technology
(1974)

M.Sc. Northwestern University
(1976)

SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE
DEGREE OF

DOCTOR OF PHILOSOPHY

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

(August, 1979)

© Chat-Yu Lam 1979

Signature of Author ....
Sloan School of Management, August 1, 1979

Certified by ....
Thesis Supervisor

Accepted by ....
Chairman, Department Committee

Report Documentation Page

Form Approved
OMB No. 0704-0188

Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.

1. REPORT DATE
AUG 1979

2. REPORT TYPE

3. DATES COVERED
00-00-1979 to 00-00-1979

4. TITLE AND SUBTITLE
Data Storage Hierarchy Systems for Data Base Computers

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Massachusetts Institute of Technology,77 Massachusetts Avenue,Cambridge,MA,02139

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR’S ACRONYM(S)

11. SPONSOR/MONITOR’S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The need for efficient storage and processing of very large databases to support decision-making coupled with advances in computer hardware and software technology have made research and development of specialized architectures for database management a very attractive and important area. The INFOPLEX data base computer proposed by Madnick applies the theory of hierarchical decomposition to obtain a specialized architecture for database management with substantial improvements in performance and reliability over conventional architectures. The storage subsystem of INFOPLEX is realized using a data storage hierarchy. A data storage hierarchy is a storage subsystem designed specifically for managing the storage and retrieval of very large databases using storage devices with different cost/performance characteristics arranged in a hierarchy. It makes use of locality of data references to realize a low cost storage subsystem with very large capacity and small access time. As part of the INFOPLEX research effort, this thesis is focused on the study of high performance, highly reliable data storage hierarchy systems. Concepts of the INFOPLEX data base computer are refined and new concepts of data storage hierarchy systems are developed. A preliminary design of a general structure for the INFOPLEX data storage hierarchy system is proposed. Theories of data storage hierarchy systems are developed. Madnick’s model of a generalized storage hierarchy is extended and formalized for data storage hierarchy systems. The Least Recently Used (LRU) algorithm is extended to incorporate the read-through strategy and page overflow strategies to obtain four classes of data movement algorithms. These algorithms are formally defined. Important performance and reliability properties of data storage hierarchy systems that make use of these algorithms are identified and analyzed in detail. It is proved in Theorems 1 and 2 that depending on the relative sizes of the storage levels and the algorithms used, it is not always possible to guarantee that the contents of a given storage level ’i’ is a superset of the contents of its immediate higher storage level ’i-l’, i.e., multi-level inclusion (MLI) does not hold. Necessary and sufficient conditions for MLI to hold are identified and proven in Theorems 3 and 4. A property related to MLI is the multi-level overflow inclusion (MLOI) property. MLOI holds if an overflow page from storage level ’i’ is always found to already exist in storage level ’i+l’. A data storage hierarchy avoids cascaded references to lower storage levels if MLOI holds. Necessary and sufficient conditions for the MLOI to hold are identified and proven in Theorems 5 and 6. It is possible that increasing the sizes of intermediate storage levels may

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**320**	

Data Storage Hierarchy Systems for
Data Base Computers

by
Chat-Yu Lam

Submitted to the Alfred P. Sloan School of Management
on August 1979 in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

ABSTRACT

The need for efficient storage and processing of very large databases to support decision-making coupled with advances in computer hardware and software technology have made research and development of specialized architectures for database management a very attractive and important area.

The INFOPLEX data base computer proposed by Madnick applies the theory of hierarchical decomposition to obtain a specialized architecture for database management with substantial improvements in performance and reliability over conventional architectures. The storage subsystem of INFOPLEX is realized using a data storage hierarchy. A data storage hierarchy is a storage subsystem designed specifically for managing the storage and retrieval of very large databases using storage devices with different cost/performance characteristics arranged in a hierarchy. It makes use of locality of data references to realize a low cost storage subsystem with very large capacity and small access time.

As part of the INFOPLEX research effort, this thesis is focused on the study of high performance, highly reliable data storage hierarchy systems. Concepts of the INFOPLEX data base computer are refined and new concepts of data storage hierarchy systems are developed. A preliminary design of a general structure for the INFOPLEX data storage hierarchy system is proposed.

Theories of data storage hierarchy systems are developed. Madnick's model of a generalized storage hierarchy is extended and formalized for data storage hierarchy systems. The Least Recently Used (LRU) algorithm is extended to incorporate the read-through strategy and page overflow strategies to obtain four classes of data movement algorithms. These algorithms are formally defined. Important performance and reliability properties of data storage hierarchy systems that make use of these algorithms are identified and analyzed in detail. It is proved in Theorems 1 and 2 that depending on the relative sizes of the storage levels and the algorithms used, it is not always possible to guarantee that the contents of a given storage level 'i' is a superset of the contents of its immediate higher storage level 'i-1', i.e., multi-level inclusion (MLI) does not hold. Necessary and sufficient conditions for MLI to hold are identified and proven in Theorems 3 and 4. A property related to MLI is the multi-level overflow inclusion (MLOI) property. MLOI holds if an overflow page from storage level 'i' is always found to already exist in storage level 'i+1'. A data storage hierarchy avoids cascaded references to lower storage levels if MLOI holds. Necessary and sufficient conditions for the MLOI to hold are identified and proven in Theorems 5 and 6. It is possible that increasing the sizes of intermediate storage levels may actually increase the number of references to lower storage levels, resulting in decreased performance. This is referred to as the multi-level paging anomaly (MLPA). Conditions necessary to avoid MLPA are identified and proven in Theorems 7 and 8.

A simplified structure of the INFOPLEX data storage hierarchy is derived from its general structure. Protocols for supporting the read-through and store-behind algorithms are specified. Two simulation models of this system are developed. The first model incorporates one functional processor and three storage levels. Results from this model provide significant insights to the design and its algorithms and reveals a potential deadlock in the buffer management schemes. The second model corrects this potential deadlock and also incroproates five functional processors and four storage levels. Results from this model show that the store-behind operation may be a significant system bottleneck because of the multi-level inclusion requirement of the data storage hierarchy. By using more parallelism in the lower storage levels and by using smaller block sizes it is possible to obtain a well-balanced system which is capable of supporting the storage references generated by the INFOPLEX functional hierarchy. The effects of using projected 1985 technology for the data storage hierarchy are also assessed.

Thesis Supervisor: Prof. Stuart E. Madnick
Associate Professor of Management Science
M.I.T. Sloan School of Management

ACKNOWLEDGMENT

I would like to sincerely thank the following individuals who helped and encouraged me in carrying out this work.

My thesis advisor, Prof. Stuart Madnick, has been a great teacher and a concerned and understanding advisor. He has spent considerable time and effort supervising this thesis. It is a priviledge to be his student.

Members of my thesis committee, Prof. John Little, Prof. Hoo-min Toong, and Dr. Ugo Gagliardi, have taken time in spite of their tight schedules to review this work. I am truely grateful for their interest and enthusiasm.

My colleagues, Sid Huff and Mike Abraham, have at various times suffered through some of my ill-formed ideas. Their patience is greatly appreciated.

My wife, Elizabeth, has stood by me throughout this work with her patience, encouragements and care. She makes all this worthwhile.

I would like to acknowledge the sources of funding for this work, and for my doctoral studies.

This research is funded by the National Science Foundation Grant No. MCS77-20829 and by the Navy Electronics Systems Command Contract No. N00039-78-G-0160.

My doctoral studies have been made possible by the above fundings, several Sloan School tuition fellowships, an IBM Information Systems Fellowship, and research funds from the NEEMIS Project and the RADC Decision Support Systems Project. I am grateful for these financial support.

A large portion of this thesis was typed by Waterloo SCRIPT. Chapter 4 was typed by Marilyn Whatmough and Portia Smith who have been kind enough to undertake this task that nobody else would, due to the large number of mathematical symbols in the chapter.

TABLE OF CONTENTS

Appendix page

Chapter I

INTRODUCTION AND PLAN OF THESIS

1.1 INTRODUCTION

The effective and efficient storage and processing of very large data bases to support better decision-making has been a major concern of modern organizations. Though advances in computer technology are impressive, the rate of growth of information processing in organizations is increasing even more rapidly. A key technological advance in providing better information processing systems is the development of Data Base Management Systems (DBMS's) (Martin, 1975). Most organizations today make use of some kind of DBMS for handling their large databases. Efforts to develop even more effective DBMS's remain very active and important (Mohan, 1978).

Current DBMS's are capable of handling large databases on the order of trillion bits of data (Simonson and Alsbrooks, 1975), and are capable of handling query rates of up to one hundred queries per second (Abe, 1977). Due to the increasing need for better information, and the declining costs of processors and storage devices, it is expected that future high performance DBMS's will be required to handle query rates and provide storage capacities several orders of magnitude higher than today's (Madnick, 1977). Furthermore, with such high query rates (generated by terminal users as well as directly from other computers), it is essential that a DBMS maintains non-stop operation (Computerworld, 1976). Thus, guaranteeing the reliability of the DBMS becomes very difficult.

Current improvements in processor and storage device technology alone do not seem to be able to meet these orders of magnitude improvements in performance and reliability. The nex section reviews several research efforts aimed at modifying the conventional computer architecture for better information handling. One such research effort is the INFOPLEX Project (Lam and Madnick, 1979). The INFOPLEX approach to obtaining a high performance, highly reliable DBMS is to design a new computer specifically for data management. This thesis is a study of the storage subsystem of the INFOPLEX data base computer. Research goals and specific accomplishments of this thesis will be described in a following section. The structure of this thesis is then described.

1.2 RELATED RESEARCH

In the past, computers were designed primarily for numerical computation. We now find that processing of large databases has become a major, if not dominant, component of computer usage. However, current computer structures still have the 'von Neumann' structure of twenty years ago. As Mueller (Mueller, 1976), President of System Development Corporation, noted: 'the computer industry has gone through three generations of development to perfect machines optimized for 10 percent of the workload'. It is not surprising then, that many organizations find their supercomputers running 'out of steam' as new applications with large databases are installed.

Figure 1.1 illustrates a simplified typical computer architecture. It consists of a processor directly accessing a main memory (with access time in the order of microseconds), an I/O processor that controls the movement of data between main memory and secondary storage devices, an I/O controller and its associated secondary storage devices (with access times in the order of milliseconds). Current DBMS's are software systems that reside in the main memory together with other software subsystems and application programs. To provide a high level view of data for application programs a DBMS has to manage all the data residing in sec-

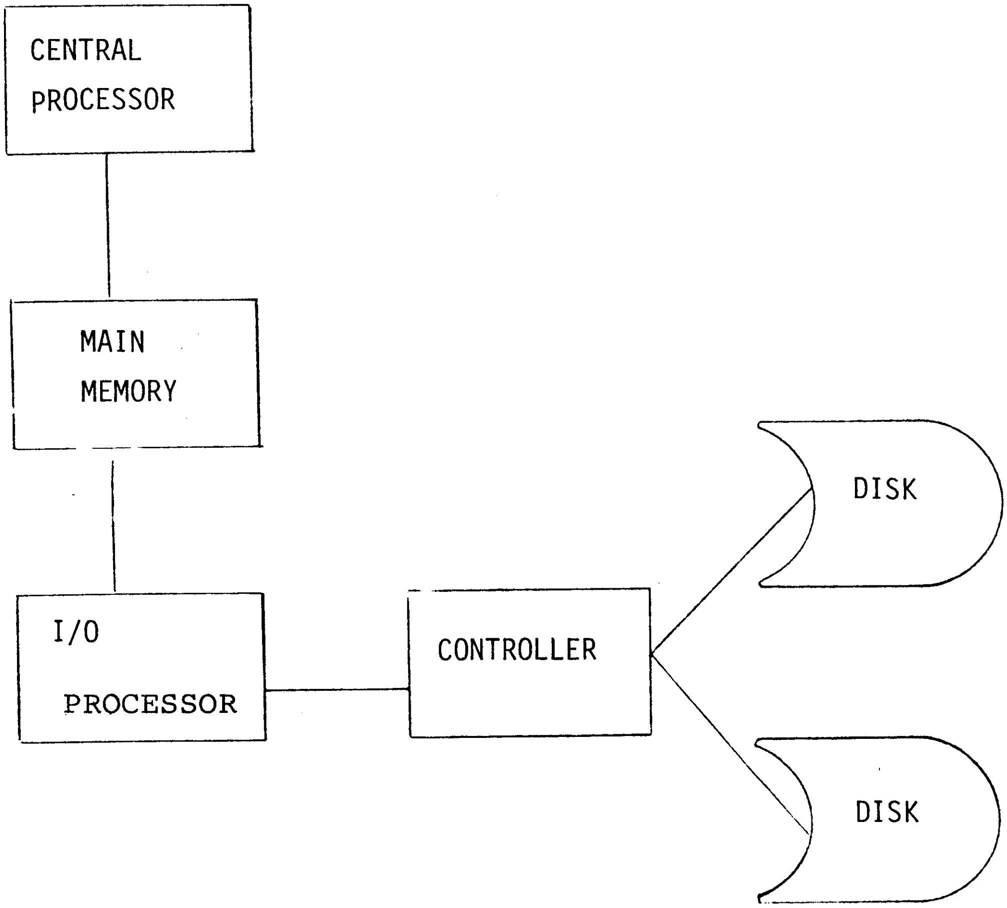


Figure 1.1 A Typical Computer Architecture

ondary storage devices and coordinate all the data movement and processing activities. Two potential deficiencies of adapting the conventional computer architecture for data base management become evident. First, the processor becomes strained as new functions are added to the DBMS. These new functions include high level language support, better security and data integrity mechanisms, support of multiple data models, ..., and so on. Second, due to the large differential in access times of main memory and secondary storage devices (referred to as the 'access gap'), the speed of processing becomes limited by how fast useful data can be brought into main memory from secondary storage devices. Thus, many organizations find the performance of their data management system either limited by the available processor cycles or limited by the speed of I/O operations, depending on the DBMS used and the applications supported.

These problems have been recognized for some time. Current advances in LSI technology make it feasible to consider new software and hardware architectures to overcome the above deficiencies. Several such approaches are discussed below.

1.2.1 New Instructions Through Microprogramming

Conventional processor instructions are usually not well suited to the requirements of database management systems. Using firmware, it is possible to augment or enhance the instructions thus effectively increase the efficiency of the processor. This approach has been adopted in several systems. One of the earliest efforts occurred as part of the LISTAR information retrieval system developed at M.I.T.'s Lincoln Laboratory (Armenti *et al.*, 1970), where several frequently used operations, such as a generalized List Search operation, were incorporated into the microcode of an IBM System/360 Model 67 computer. The Honeywell H60/64 uses special instructions to perform data format conversion and hashing corresponding to frequently used subroutines of Honeywell's IDS database system (Bachman, 1975). More recently the IBM System/38 (Soltis and Hoffman, 1979) was announced with microcode to perform much of the operating system and data management functions. The performance advantages of this approach are highly dependent upon the frequency of use of the new instructions and the extent to which they fit into the design of the overall database system software.

1.2.2 Storage Hierarchy Optimization

It is possible to close the 'access gap' between main memory and secondary storage devices by using a more continuous storage hierarchy, thus improving the performance of the storage subsystem.

Madnick (Madnick, 1973) proposed a model of a generalized storage hierarchy system and its data movement algorithms. This storage hierarchy makes use of multiple page sizes across the storage levels for high performance and maintains multiple data redundancy across the storage levels for high performance and high reliability. This type of storage hierarchy systems have great potentials as storage subsystems for high performance, highly reliable DBMS's. Unfortunately, the lack of better understanding of this type of storage hierarchy systems is a major obstacle in the development of practical storage subsystems in spite of the fact that a continuous spectrum of storage devices with different cost/performance characteristics will persist (Dennis _et_. _al_., 1978; Hoagland, 1979; Smith, 1978a).

There has been much work on studying storage hierarchy systems and their algorithms. We shall review these work in a later chapter. These studies usually do not consider the effects of multiple page sizes across different storage levels, nor the problems of providing multiple data redundancy across the storage levels, as in the system proposed by Madnick. Developing theories for generalized storage hierarchy systems specifically for managing large database remains a challenge.

1.2.3 Intelligent Controllers

Another approach to improving information processing efficiency is to use intelligent controllers. The controller provides an interface between the main memory and the devices. Recently, more and more intelligence has been introduced into these controllers. For example, many controllers can perform the search key operation themselves (Ahern _et al._, 1972; Lang _et al._, 1977). Since only selected data items are brought to the main memory, the I/O traffic is reduced and the efficiency of the storage subsystem is increased.

Two major types of intelligent controllers have emerged. The first type specializes in automating the data transfer between the storage devices, i.e., the physical storage management functions. For example, IBM's 385Ø Mass Storage System (Johnson, 1975) uses an intelligent controller to automatically transfer data between high-capacity, slow-speed tape cartridges and medium-capacity, fast moving-head disks. Thus, the processor is relieved of the burden to manage these data movements.

The second type of intelligent controllers is designed to handle some of the logical storage management functions, such as searching for a specific data record based on a key. This latter type of device is sometimes referred to as a database computer, and is often used to perform associative or parallel searching (Langdon, 1978). Most parallel associative search strategies are based on a head-per-track storage device technology (for example, magnetic drums, LSI shift registers, and magnetic bubbles) and a multitude of comparators. As each data record rotates, either mechanically or electronically, past a read/write head, it is compared with a match record register, called the mask. Examples of this type of intelligent controllers include CASSM (Copeland et al., 1973; Healy et al., 1972; Su and Lipovski, 1975; Su, 1977; Su et. al., 1979), the Rotating Associative Relational Storage (RARES) design (Lin et al., 1976), and the Rotating Associative Processor (RAP) (Ozkarahan et al., 1975; Schuster et al., 1976; Ozkarahan et al., 1977; Schuster, 1978; Schuster, et. al., 1979).

Although the decline in the costs of comparator electronics, due to advances in LSI technology, makes parallel search strategies quite promising for the future, they are only well suited to storage technologies that lend themselves to low cost read/write mechanisms, and for optimal performance and operation they tend to require a fairly simple and uniform database structure (e.g., relational flat files). To use these intelligent controllers in conjunction with other storage devices, such as mass storage, some "staging" mechanisms have to be used. Furthermore, these intelligent controllers only support part of the information management functions, much of the complex functions of language interpretation, support of multiple user interfaces, etc., of an information management system cannot easily be performed in these controllers.

1.2.4 Back-end Processors

The fourth approach is to shift the entire database management function from the main computer to a dedicated computer thus increasing the processor power available for performing the data management function. Such a computer is often called a back-end processor. The back-end processor is usually a minicomputer specifically programmed to perform all of the functions of the database management system.

Back-end processors have evolved rapidly in recent years. Some of the earliest experimental efforts include the loosely coupled DATACOMPUTER (Marill and Stern, 1975), developed by the Computer Corporation of America using the DECSystem-1Ø computer, and the tightly coupled XDMS (Canady et al., 1974), developed by Bell Laboratories by modifying the firmware of a Digital Scientific META-4 minicomputer. More recent developments include the Cullinane Corporation's IDMS on a PDP/11 compuater. Since the back-end processor is still a conventional computer whose architecture has been designed for computational purposes, not for information management, its performance is still quite limited.

1.2.5 Data Base Computers

The fifth approach to providing improved information processing efficiency is the database computer. The difference between this approach and the fourth approach (back-end processor) is that the database computer has a system architecture particularly suitable for database operations while a back-end processor merely adapts a conventional computer to database applications.

There has been relatively little research on the development of true database computers (as opposed to work on intelligent controllers and/or dedicated back-end processors -- which are sometimes referred to as database computers). Current data base computer research efforts include the DBC (Hsiao and Kannan, 1976; Banerjee, et. al., 1978; Banerjee, et. al., 1979) at the Ohio State University, the GDS (Hakozaki et al., 1977) at the Nippon Electric Co., Japan, and the INFOPLEX effort at M.I.T. (Madnick, 1975b; Lam and Madnick, 1979; Madnick, 1979).

Data Base Computer seems to be a long term solution to the DBMS requirements of future computer systems. The DBC approach at Ohio State University makes use of specialized functional processors for performing the data management functions thus eliminating the processor bottleneck that exists in current DBMS's. To improve the efficiency of the storage subsystem, the DBC makes use of the idea of a partitioned content addressable memory (PCAM). The entire address space is divided into partitions, each of which is content addressable. To realize content addressability cost effectively, the DBC makes use of multiple intelligent controllers at the secondary storage devices.

The INFOPLEX architecture also makes use of multiple functional processors. However, to obtain a flexible, high performance, and highly reliable storage subsystem, INFOPLEX makes use of a storage hierarchy system based on the Madnick proposal (Madnick, 1973). Conceptually, the INFOPLEX database computer consists of a functional hierarchy and a physical (storage) hierarchy (See Figure 1.2). The INFOPLEX functional hierarchy is a hierarchy of specialized microprocessors. It implements all the information management functions of a database manager, such as query language

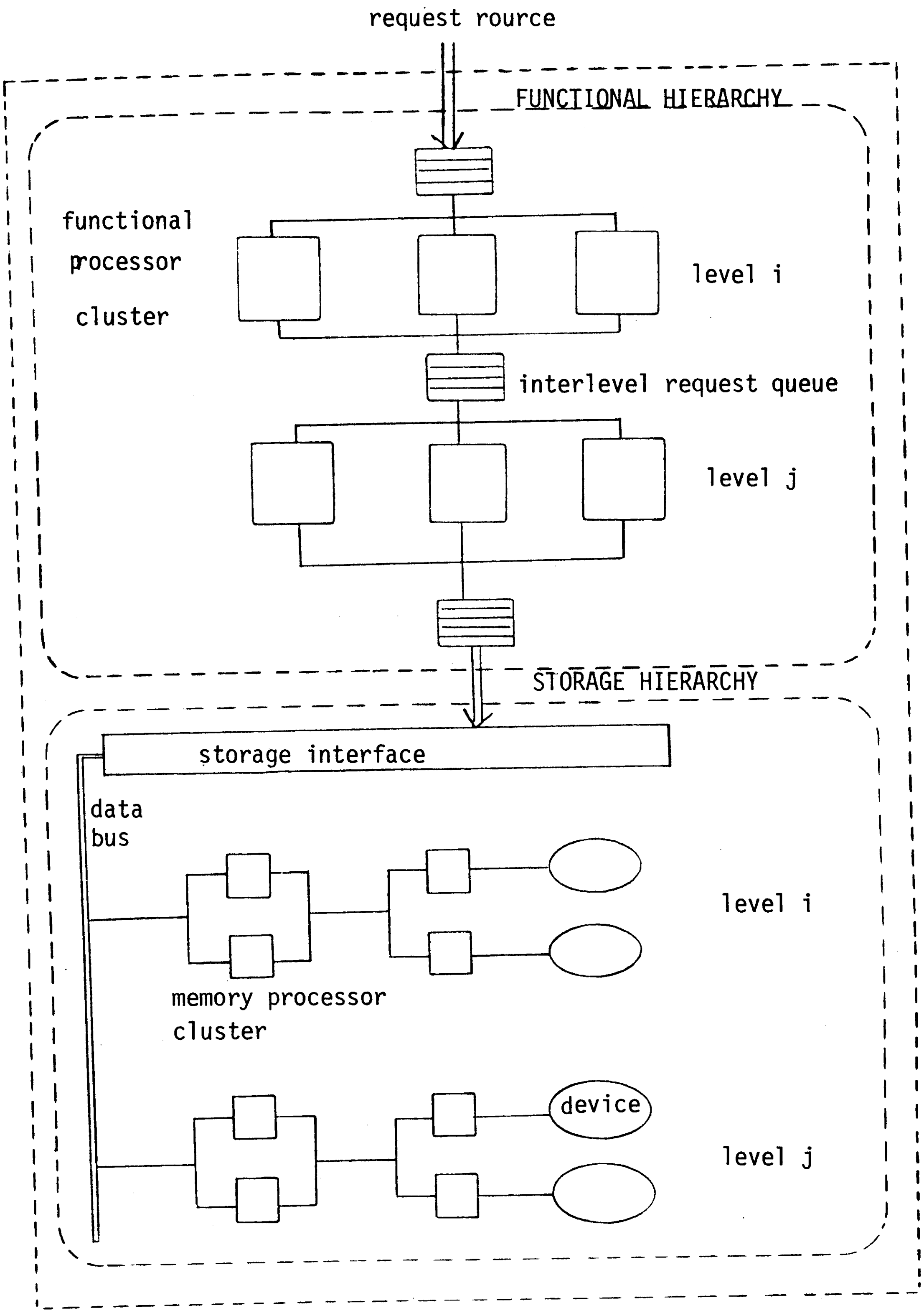


Figure 1.2 The INFOPLEX Structure

interpretation, security verification, and data path accessing, etc. The hierarchy of functional processors establishes a pipeline. Within each stage of the pipeline, multiple processors are used to realize parallel processing and provide multiple redundancy. The INFOPLEX storage hierarchy is designed specifically to support the data storage requirements of the functional hierarchy. To provide high performance and high reliability, it makes use of a highly parallel and reliable architecture, implements distributed control mechanisms, and maintains multiple data redundancy.

1.3 RESEARCH GOALS AND ACCOMPLISHMENTS

This thesis is a study of the INFOPLEX data storage hierarchy. We have studied data storage hierarchy systems from five different and related points of view: (1) development of concepts for INFOPLEX and data storage hierarchy systems, (2) architectural design of data storage hierarchy systems, (3) theoretic analysis of data storage hierarchy systems, (4) algorithm development for data storage hierarchy systems, and (5) performance evaluation of data storage hierarchy systems. Specific goals and accomplishments of this thesis are listed below.

1. Develop and extend concepts of data base computers and data storage hierarchy systems: Since Madnick's (Madnick,1975b) proposal to develop a high performance, highly reliable data base computer, called INFOPLEX, there has been many alternative approaches to develop special architectures for data base management. We have reviewed these proposals and categorized these efforts into: (1) new instructions through microprogramming, (2) storage hierarchy optimization, (3) intelligent controllers, (4) back-end processor, and (5) data base computers. Concepts of the INFOPLEX data base computer have been refined and leads to the development of the concept of a data storage hierarchy.

2. Architectural design of data storage hierarchy systems: Although storage hierarchy systems with two or three levels are very common in current computer systems, there is no known storage hierarchy with more than three storage levels that has been designed specifically for large databases. A preliminary design of the general structure of a data storage hierarchy with an arbitrary number of storage levels has been developed. This structure

is the basis for future designs of data storage hierarchy systems for the INFOPLEX data base computer.

3. Theoretic analysis of data storage hierarchy systems: Madnick (Madnick, 1973) proposed the model of a generalized storage hierarchy system that incorporates multiple page sizes and maintains multiple data redundancy for high performance and high reliability. This model is extended and formalized for data storage hierarchy systems. The Least Recently Used (LRU) algorithm is extended to incorporate the read-through strategy for managing the data movement in data storage hierarchy systems. Four classes of algorithms are obtained and formally defined. The multi-level inclusion (MLI), multi-level overflow inclusion (MLOI), and multi-level paging anomaly (MLPA) properties of data storage hierarchy systems using these algorithms are analyzed in detail and formally proved as eight theorems and nine lemmas.

4. Develop algorithms for data storage hierarchy systems: A simplified structure of the INFOPLEX data storage hierarchy is obtained from the general structure. Protocols to support the read-through

and the store-behind data movement algorithms are developed for this structure.

5. Performance evaluation of data storage hierarchy systems: Two GPSS/360 simulation models of the INFOPLEX data storage hierarchy are developed. Simulation results reveal several unexpected properties of the data storage hierarchy design and its algorithms. A well-balanced system is used to compare the performance differential of using technology in 1979 versus projected technology in 1985. These simulation results indicate that the current INFOPLEX data storage hierarchy design is capable of supporting the read and write traffic generated by the INFOPLEX functional hierarchy.

1.4 STRUCTURE OF THESIS

This thesis is an important step towards developing storage hierarchy systems specifically for data base computers. Existing models of storage hierarchy systems are extended to obtain a formal model of storage hierarchy system which incorporates multiple page sizes and maintains multiple data redundancy. Key properties of such systems are analyzed in detail. Architectures of storage hierarchy systems for INFOPLEX are developed and the performance of these designs are evaluated. Details of this research are presented in seven chapters. Chapter one is self-explainatory. The following outlines the contents of the other chapters.

1.4.1 Chapter 2 : The INFOPLEX Data Base Computer Architecture

This chapter introduces the objectives and approaches of the INFOPLEX data base computer. Concepts and research approaches used in the INFOPLEX functional hierarchy and the INFOPLEX data storage hierarchy are described. This chapter provides the background and motivation for the research on data storage hierarchy systems.

1.4.2 Chapter 3 : A General Structure of the INFOPLEX Data Storage Hierarchy

A preliminary design of the INFOPLEX data storage hierarchy, DSH-1, is proposed. The design objectives of DSH-1 are discussed. Then the structure of DSH-1 is introduced. This design can be used to explore design issues associated with the INFOPLEX data storage hierarchy. Key design issues are identified.

1.4.3 Chapter 4 : Modelling and Analysis of Data Storage Hierarchy Systems

Current research efforts in storage hierarchy systems are briefly reviewed. A formal model of data storage hierarchy systems incorporating multiple page sizes and maintain multiple data redundancy is developed. Extensions to the Least Recently Used (LRU) algorithm are developed to incorporate the read-through strategy. Important performance and reliability properties of these systems are formally proved. These results provide valuable insights to designing data storage hierarchy systems. The formalisms developed provide a solid basis for further theoretic analysis of data storage hierarchy systems.

1.4.4 Chapter 5 : Design of the DSH-11 Data Storage Hierarchy System

The general structure of the INFOPLEX data storage hierarchy is used to derive a simpler structure, DSH-11. This structure is used as a basis for developing protocols for supporting the read and write operations. Specifications for these protocols are presented.

1.4.5 Chapter 6 : Simulation Studies of the DSH-11 Data Storage Hierarchy System

A simulation model of DSH-11 with one processor and three storage levels is developed. Results from simulation studies using this model provide valuable insights to the DSH-11 design and its algorithms. This knowledge is incorporated into another simulation model of DSH-11 that consists of five processors and four storage levels. Simulation studies from this model reveal further interesting properties of the read-through and store-behind algorithms. The simulation results also indicate that the current design is capable of supporting the very high rate of storage references generated by the INFOPLEX functional hierarchy.

1.4.6 Chapter 7 : Discussions and Conclusions

Chapter 7 summarizes this thesis and indicates fruitful areas for further research.

Chapter II

THE INFOPLEX DATA BASE COMPUTER ARCHITECTURE

2.1 INTRODUCTION

This chapter discusses the INFOPLEX data base computer concepts and its approaches. Specific areas of contribution of this thesis to the development of the INFOPLEX data base computer are then listed.

The key concepts of the INFOPLEX architecture are hierarchical decomposition and distributed control. Techniques of hierarchical decomposition are applied to organize the information management functions to obtain a highly modular functional hierarchy. Each level of the functional hierarchy is implemented using multiple microprocessors. Techniques of hierarchical decomposition are also applied to organize the storage subsystem to obtain a modular storage hierarchy capable of supporting the storage requirements of the functional hierarchy. Microprocessors are used at each level of the hierarchy to implement the storage management algorithms so the hierarchy appears as a very large, highly reliable, high performance virtual storage space.

Due to the high modularity of these organizations, both the functional hierarchy and the storage hierarchy can take advantage of distributed control mechanisms. Each level in a hierarchy only communicates with its adjacent levels and each module within a level only communicates with its adjacent modules. Thus, no central control mechanism is necessary. Distributed control enhances reliability since there is no single component in the system whose failure renders the entire system inoperative. Distributed control also enhances performance since there is no system bottleneck as would exist in a centrally controlled system.

A functionally decomposed hierarchy, implemented using multiple microprocessors, can support pipeline processing naturally. That is, multiple requests for information can be at various stages of processing at different levels of the hierarchy simultaneously. Such an architecture also enhances reliability since errors can be isolated within a level in the hierarchy thus simplifying error detection and correction.

Parallel processing is made possible by the hierarchical decomposition and implementation using multiple microprocessors. For example, there may be several identical modules that implement the same function within a level. All these modules can be simultaneously operating on different requests, at the same time, providing potential backup for one another.

Thus, the distributed control, pipeline and parallel processing capabilities of INFOPLEX provide very high reliability and high performance.

In addition to providing high performance and high reliability, a viable data base computer must be able to take advantage of new technological innovations. It must be able to easily upgrade to incorporate new algorithms, e.g., a new security checking technique, or new hardware innovations, e.g., a new storage device. Due to its modular structure, the INFOPLEX functional hierarchy can take advantage of new techniques and technologies as they are developed. The INFOPLEX storage hierarchy is specifically designed to be able to handle any type of storage devices. Thus rather than being specialized to a particular data structure, or type of storage device, INFOPLEX is designed to adapt to the changing application needs as well as to take advantage of new technological innovations.

2.2 THE INFOPLEX FUNCTIONAL HIERARCHY

An information management system performs a spectrum of very complex functions in response to user requests for information. These requests are often expressed using very high level languages and often come from many different sources simultaneously. There are many ways that these complex functions can be implemented. The technique of _hierarchical functional decomposition_ has been found to be very effective for advanced information systems (Donovan and Jacoby, 1975). Similar techniques have been used successfully in operating systems (Dijkstra, 1968; Madnick and Donovan, 1974), basic file systems (Madnick and Alsop, 1969; Madnick, 1970), and a wide range of complex systems (Pattee, 1973).

This is the approach used in INFOPLEX. The information management functions are systematically decomposed into a functional hierarchy, referred to as the INFOPLEX functional decomposition. The functional modules in the hierarchy are then implemented using multiple microprocessors.

2.2.1 Rationale for Functional Decomposition

The central idea underlying the hierarchical functional decomposition approach involves decomposing the system into a hierarchy consisting of a number of levels, such that each level interacts only with the levels below it in the hierarchy. Proper selection of the hierarchy allows design or operating problems that previously impacted the entire system, to be isolated to one or a few specific hierarchical levels, and thereby more easily handled (Parnas, 1976).

Isolating the information management functions into minimally interrelated modules facilitates the use of multiple identical modules for performing the same function, so that reliability and parallelism are enhanced. Furthermore, this approach provides great flexibility in the technologies used for implementating each type of functional module. For example, a particular data structure may be selected from a spectrum of indexing techniques for a given module without affecting the design of other types of modules.

2.2.2 Example of a Functional Decomposition

To illustrate the hierarchical functional decomposition concept, a specific example of a functional decomposition is discussed in this section. Figure 2.1 illustrates a plausible hierarchical functional decomposition. Each level of the functional hierarchy is described below.

2.2.2.1 Entities and Entity Sets

At the most fundamental level, a database system stores information about things, or entities. Also, it is usually the case that entities represented in a database fall naturally into logical groups, or "entity sets". The way in which a database system (a) represents and stores information about entities themselves, and (b) represents information about the logical grouping of entities into entity sets, forms the bedrock architecture of the system.

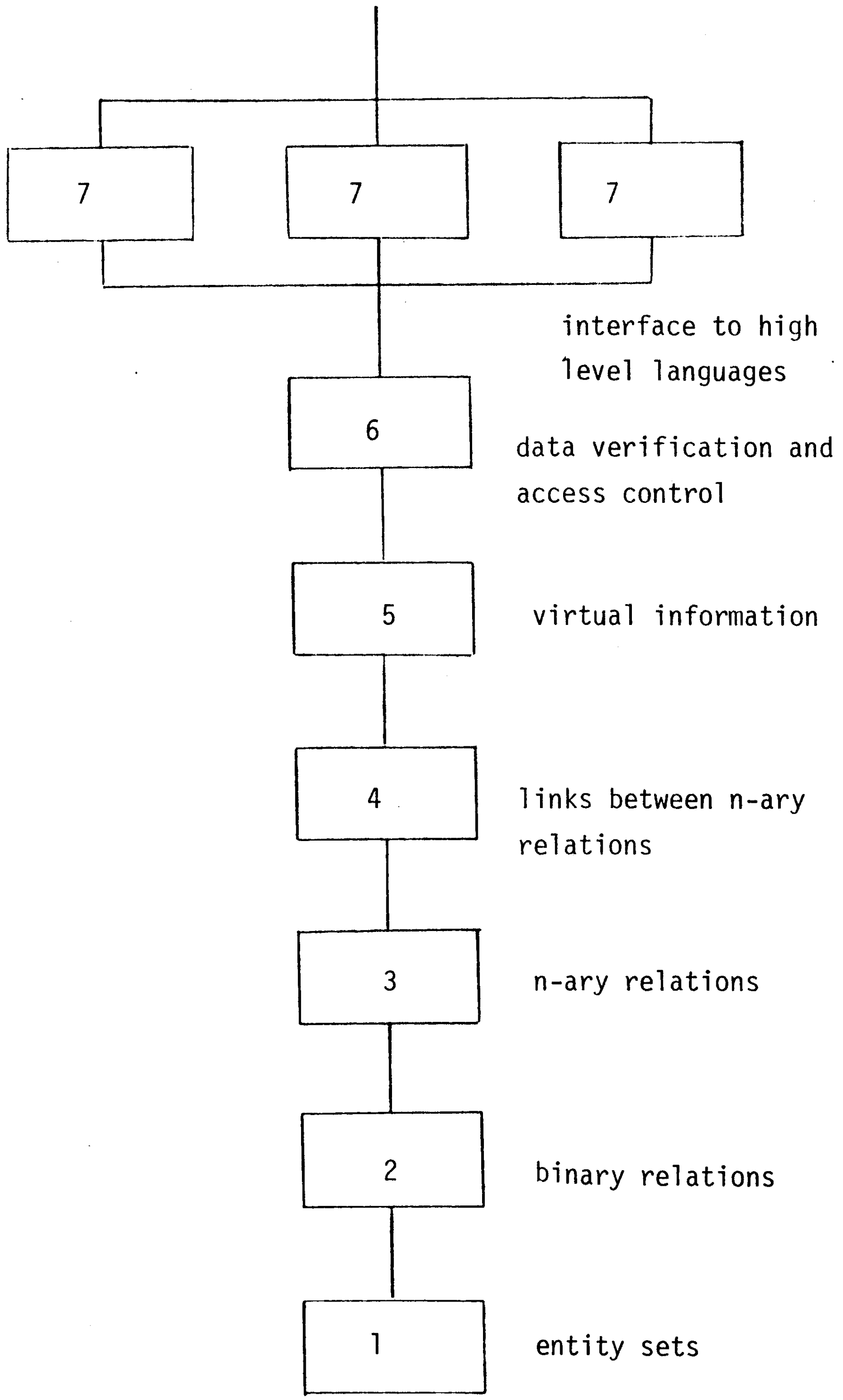

7
7
7
interface to high level languages
6
data verification and access control
5
virtual information
4
links between n-ary relations
3
n-ary relations
2
binary relations
1
entity sets

Figure 2.1 An Example Functional Hierarchy

There are many schemes available for logically and physically representing entities (i.e., coding, storing, and addressing entities) and various algorithms for structuring entity sets. The choice of implementation scheme at this level affects the performance of the entire system but does not affect how the functions of the other levels are implemented.

2.2.2.2 Binary Relations

All relationships among entities can be expressed in terms of binary relationships between pairs of entities. This functional level makes use of the entity level constructs to provide a collection of binary relations (relations between pairs of entity sets). An element of a binary relation can be viewed as a triad, consisting of a relation identifier plus two entities, each from one of the entity sets participating in the binary relation. Thus a binary relation can be viewed as a collection of triads with the same relation identifier.

Perhaps the simplest possible implementation of a set of binary relations would be as a sequential file of triads, for example,

```
(HAS_SALARY_OF ,  SMITH ,  1200)
(HAS_SALARY_OF ,  JONES ,  1500)
```

```
                ...
        (WORKS_IN_DEPT ,  SMITH ,  Ø2)
        (WORKS_IN_DEPT ,  JONES ,  Ø7)
                ...
```

The difficulties with this approach are manifest: there is great data redundancy and thus waste of storage (the relation identifiers are stored in each triad); insertion of additional triads would either have to be done out of order, or else insertions and deletions would be extremely time-consuming.

Triads could also be stored as linked lists. Alternatively hashing algorithms could be employed to locate any triad, given two of its three components. The use of linked lists can improve access speed and reduce storage requirements. On the other hand, the use of hashing algorithms would provide extremely rapid access, but would be poorer in terms of storage space utilization.

Since a database may contain billions of triads, the logical and physical structures of binary relations have serious performance implications. Many implementation schemes for binary relations are possible. Although the choice of these implementation schemes has various cost and performance implications it does not affect how the functions of the next level are implemented.

2.2.2.3 N-ary Relations

Conceptually, an n-ary relation may be thought of as a table of data, with rows of the table (usually called tuples) corresponding approximately to records in a traditional data file, and columns (or domains) corresponding to fields. Furthermore, n-ary relations may be constructed out of sets of basic binary relations. For example, the degree 4 relation EMPLOYEE_DEPT_SALARY_SEX, for which a typical entry might be

(SMITH, Ø2, 12ØØ, male),

is semantically equivalent to (i.e., contains the same information as) the three binary relations WORKS_IN_DEPT, HAS_SALARY_OF and SEX, as illustrated in Figure 2.2. We could build up n-ary relation tuples out of tuple-ids of binary relations, as illustrated in Figure 2.3. In this approach, the original data entities (SMITH, Ø1, 12ØØ, male), would be stored in permanent binary relations, and all other relations would be constructed out of binary tuple ids. Tuple ids, being uniform binary numbers, are easy and efficient to manipulate.

A number of other implementations of n-ary relations is also possible. The point is, however, that once we have an

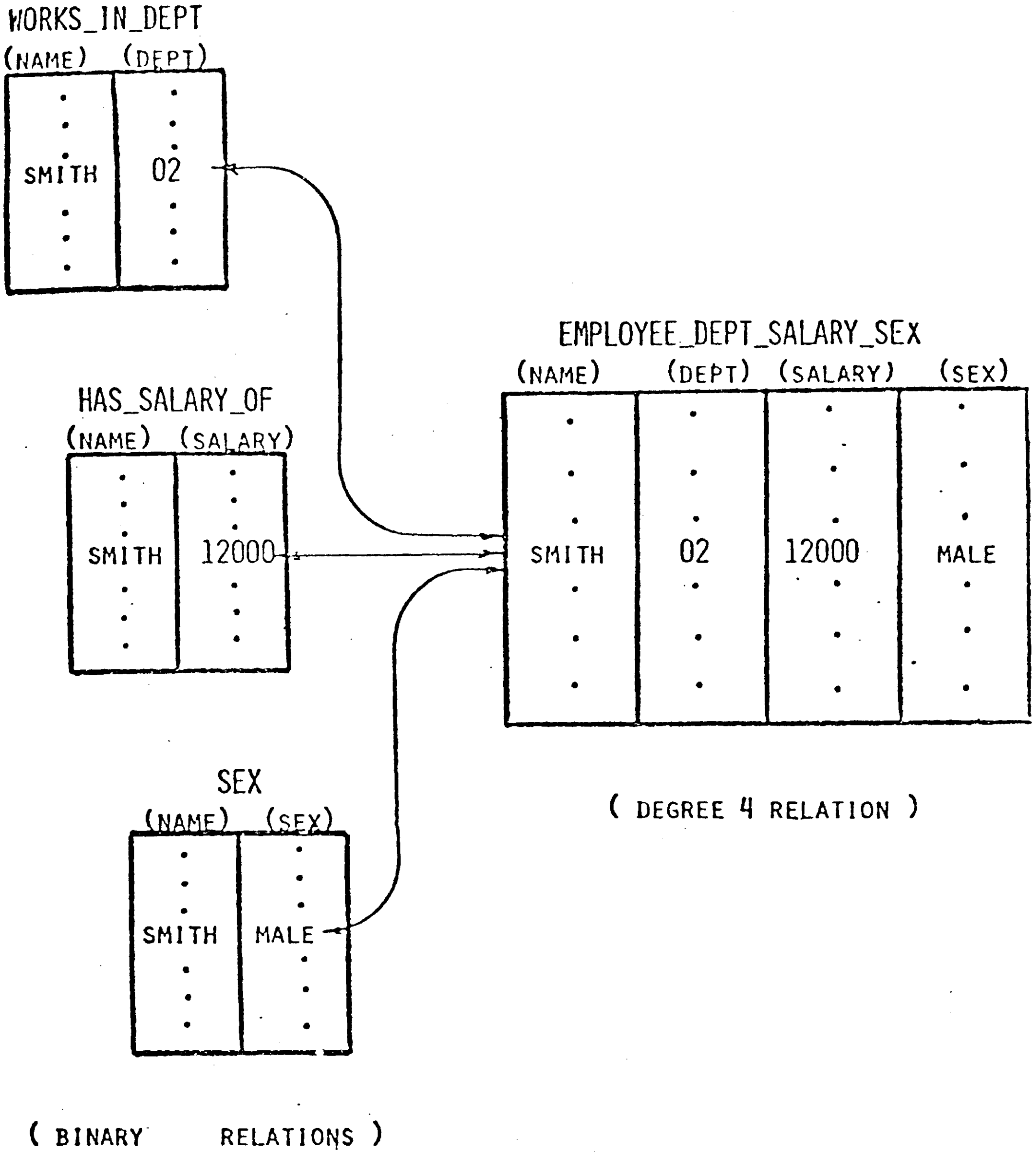


Figure 2.2 An Example 4-ary Relation

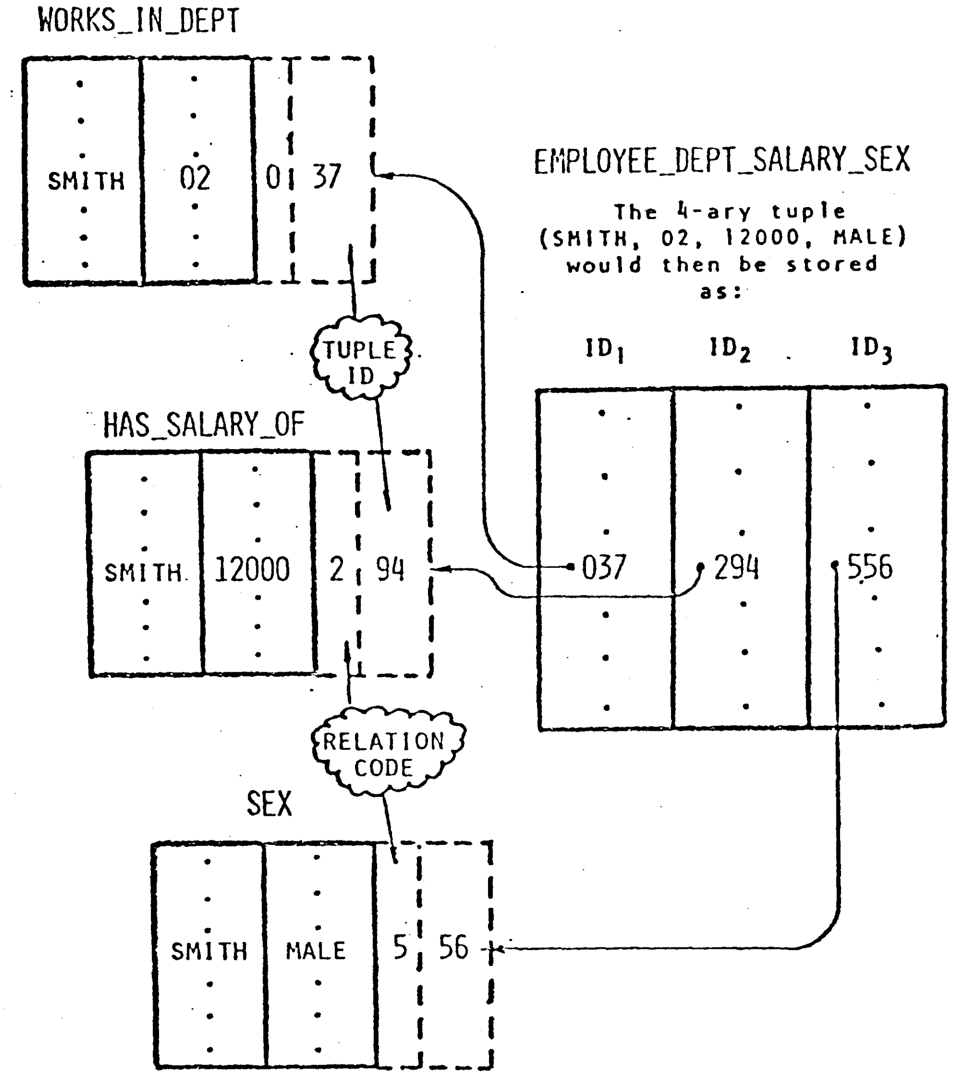


Figure 2.3 An Example Implementation of
N-ary Relations

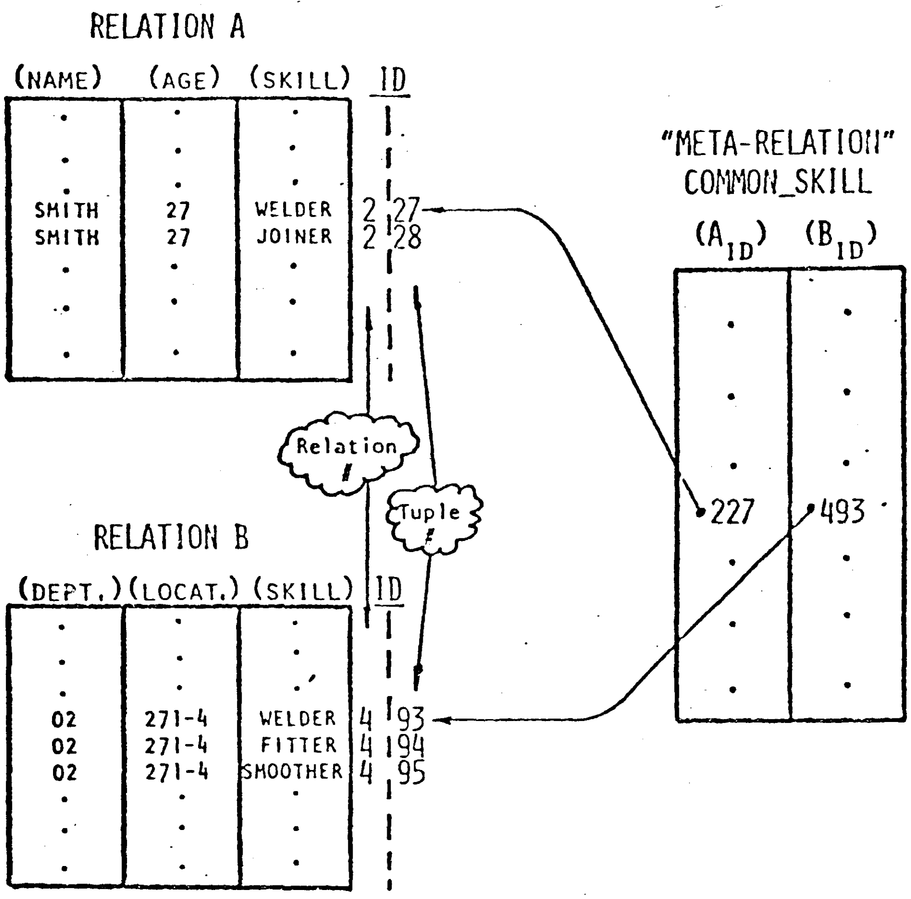


Figure 2.4 Links Among N-ary Relations

efficient implementation of binary relations, general n-ary relations may be constructed in a straightforward fashion out of the binary relations without actually having to retreat -- conceptually or physically -- back to the level of basic entities or entity sets. In other words, n-ary relation functions (to manipulate n-ary relations) can be implemented by appropriately combining binary relation functions.

2.2.2.4 Links Among N-ary Relations

The various n-ary relations in a typical database would generally possess a number of logical interconnections. For example, one relation might contain data on employees and the skills each employee possesses, while another might involve data on departments and the skills each department requires to function. The logical relationship between the tuples in these relations could be employed to extend the database structure further, by incorporating a set of "meta-relations" for storing information about such links between the regular n-ary relations. The role of the meta-relations would be to identify related tuples, and to provide some semantic information regarding the nature of the interrelationships. In the example cited above, it would make sense to establish a meta-relation connecting the appropriate tuples in the original two relations on the basis of "common skill", as shown in Figure 2.4. Under the implementation approach illustrated in Figure 2.4, meta-relations would themselves be n-ary relations. The only difference between them and regular n-ary relations lies in the interpretation of their entries. Therefore, all of the previously designed mechanisms for building and managing n-ary relations could also be used with the meta-relations. Only the interpretation of the elements within these relations would be different.

By incorporating linking information among the different n-ary relations in a database, either permanently or temporarily, directly into the database structure itself, it would be possible to generate more complex systems that would be capable of presenting different interfaces to different users, depending on the needs and objectives of the users themselves.

2.2.2.5 Virtual Information

It is not always necessary, or even desirable, that a database contain all the information that users might wish to access. Sometimes data interrelationships are algorithmic in nature, such that certain values may be unambiguously derived from others that are already stored in the database. This gives rise to the concept of "virtual" information (Folinus _et_ _al_., 1974).

If an employee's BIRTH_DATE is stored in a database, and the CURRENT_DATE is also available, then the employee's AGE could be calculated by a simple algorithm and need not also be stored. If this is in fact done, then the employee's AGE would be an example of "virtual" data -- information that appears (to the database user) to be stored there, but which is not actually present as an entity in the database.

There are a number of advantages to "virtualizing" data in a database. These include:

1. Greater accuracy: for example, an employee's AGE could be calculated as accurately as necessary if included as virtual data, whereas it would always be somewhat "old" if it were simply stored as a database entity;

2. Elimination of updating: virtual data items themselves never need updating;

3. Reduced redundancy: including, for example, BIRTH_DATE, CURRENT_DATE, and AGE as three separate items in a database is redundant, and inconsistent data relationships can easily result if some of the items are updated independently of others;

4. Savings in storage: in many cases, the database storage space required to store items such as AGE directly would be much larger than that required to store the coded algorithm for calculating AGE from other data.

One way of implementing a virtual information capability is to extend the definition of n-ary relations to include tuple identifiers ("ids") that would in fact not refer to binary relation tuples, but rather would point to procedures for calculating the virtual data items. Consider a simple employee relation of degree four, containing real data items NAME, BIRTH_DATE, and SALARY, plus a virtual data item AGE. The organization of this 4-tuple would then appear as in Figure 2.5.

2.2.2.6 Data Verification and Access Control

Data verification is the process of checking entries into a database for qualities such as reasonableness (e.g., a person's age should be no greater than, say, 125 years), and consistency (e.g., the sum of the months worked in various departments by an employee should sum to the number of months worked for the company). Access control is the process of controlling the database with regard to data retrieval, update, deletion, database reorganization, etc.

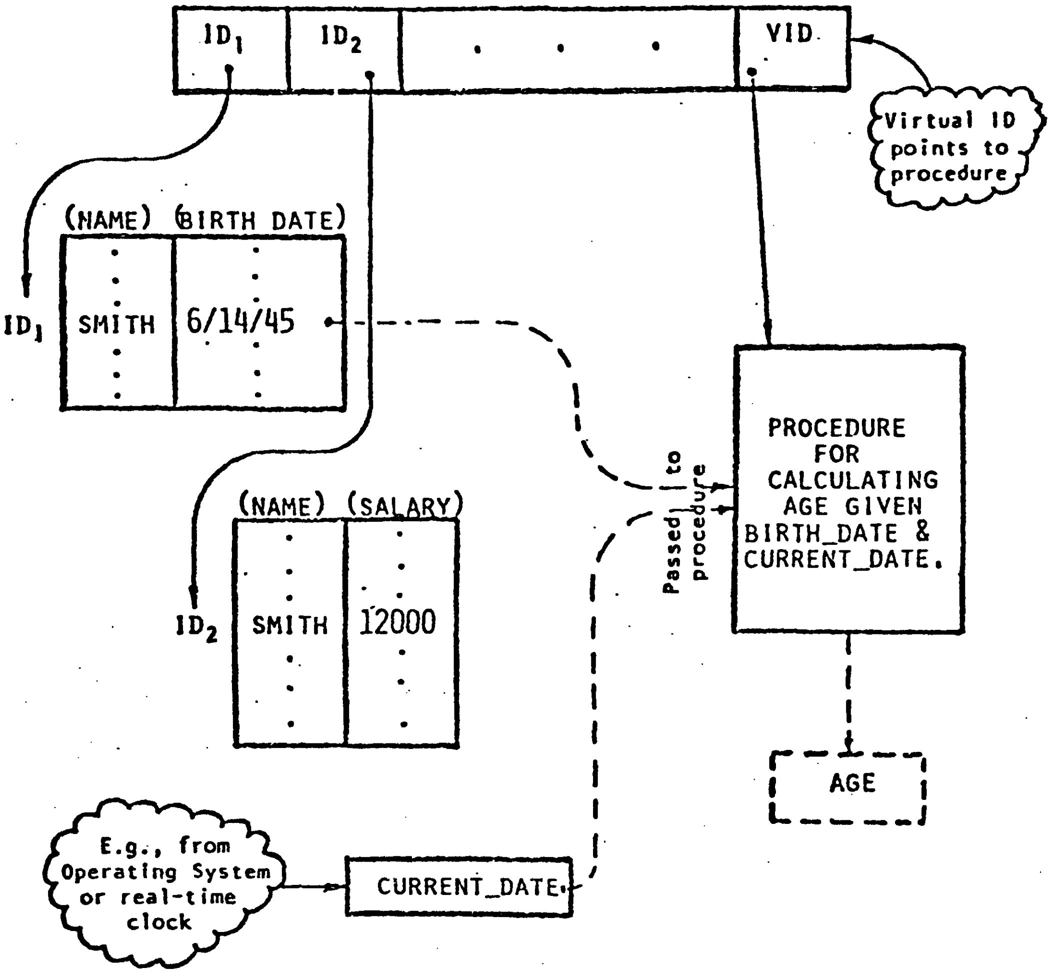

ID1
ID2
. . .
VID
Virtual ID points to procedure
(NAME) (BIRTH DATE)
ID1
SMITH
6/14/45
(NAME) (SALARY)
ID2
SMITH
12000
PROCEDURE FOR CALCULATING AGE GIVEN BIRTH_DATE & CURRENT_DATE.
Passed to procedure
AGE
E.g., from Operating System or real-time clock
CURRENT_DATE

Figure 2.5 Representation of Virtual Information

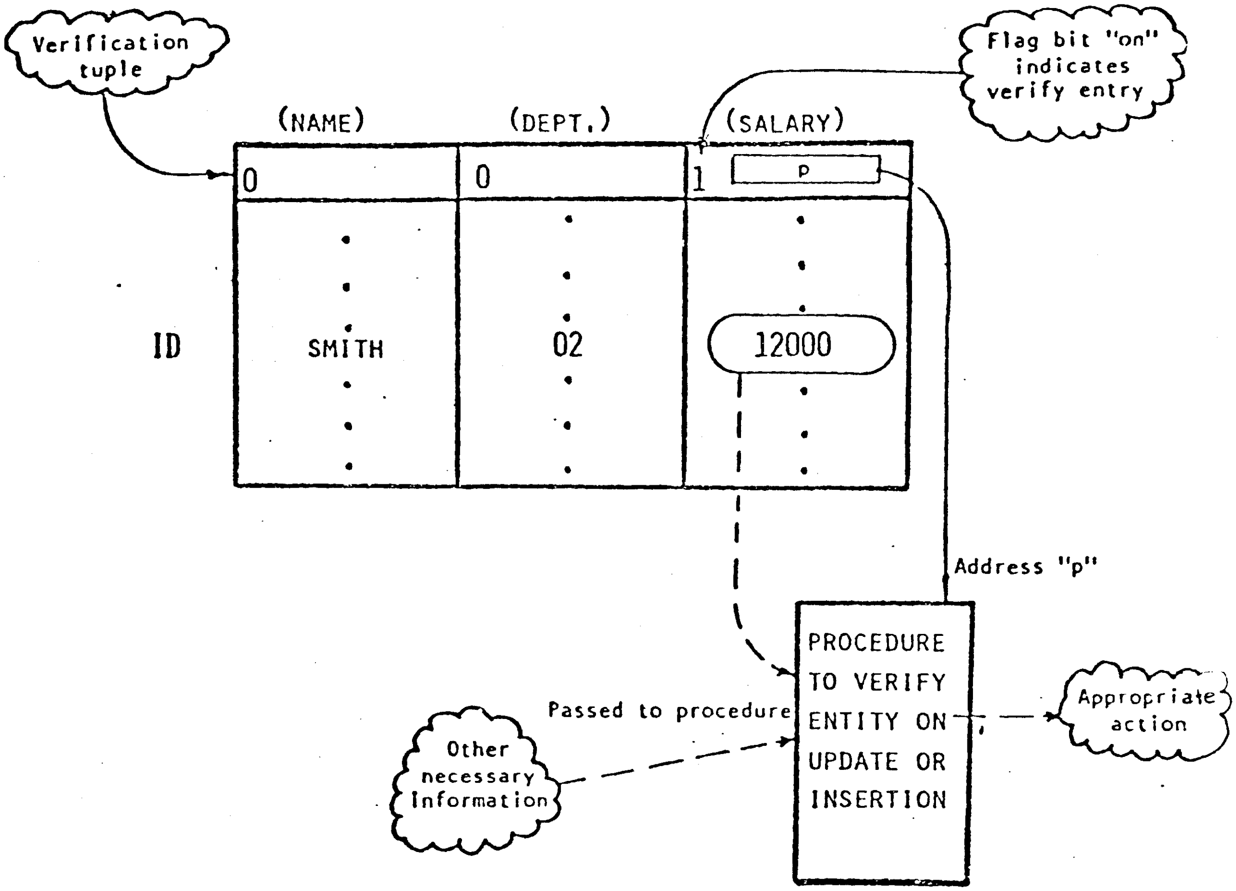

Verification tuple
Flag bit "on" indicates verify entry
(NAME)
(DEPT.)
(SALARY)
0
0
1
p
ID
SMITH
02
12000
Address "p"
PROCEDURE TO VERIFY ENTITY ON UPDATE OR INSERTION
Passed to procedure
Other necessary Information
Appropriate action

Figure 2.6 An Example Data Verification Scheme

For example, department managers may be granted authorization to view the employee records of only the employees working in their own departments; the database administrator, on the other hand, may have access to all the records in the database. The database administrator may also be the only person with authority to reorganize the entire database.

Access control also involves considerations such as the identification of valid users through use of passwords and other such techniques, mechanisms for allowing users to specify the type of access (read only, read/write, execute only, etc.) for files, and allowing users to segment files, so as to restrict parts of interconnected programs or data files from certain kinds of access by certain specified users (an example of a system that has implemented this successfully is the MULTICS system).

Both data validity and access control could be implemented in the hierarchical structure being discussed here in a variety of ways. For example, the basic n-ary relations could be further extended to include special control and verification tuples. If data verification were to be performed upon data entries in a certain domain of a relation, that domain could be flagged in a "verification tuple", and a data verification routine would be called upon data insertion or update to check the appropriateness of each entry (see Figure 2.6).

Similarly, control of access to various domains or tuples could be performed by setting control bits in a special control tuple or domain, and including, for example, an address pointer to a list of authorized user passwords, against which the current user could be checked. These control tuples or flag bits would serve to describe certain "views", or combinations of data elements, that each user would be permitted to access. Alternately, they could be used to describe elements, domains, tuples, or entire relations that a user was not permitted to view.

Note that these implementations would utilize the mechanisms employed to provide virtual information as discussed above (i.e., certain ids are used to point to verification procedures, as they pointed to "virtual information computation procedures" in the preceding section). Thus, the verification and access control functions can be realized in terms of those responsible for virtual information.

2.2.2.7 High-level Language Interface

The user interface, through the data manipulation language, basically specifies the way in which the database may be accessed by the users. In this regard, there are three main approaches to manipulating a database, corresponding roughly to the three basic models of database organization (network, hierarchical, and relational.):

1. An application programmer may wish to 'navigate' (Bachman, 1975; Codasyl, 1971) a database by using the data manipulation language to trace through the data groupings (relations) and interconnecting linkages (links between n-ary relations). This approach to database manipulation is usually more complex than some others, and demands a greater sophistication on the part of the applications programmer. He must, for example, be fully aware of the existence of all the links connecting the various data groupings, whereas this knowledge is not necessarily demanded of programmers using other data manipulation languages. In return for the greater complexity, the navigational approach usually offers greater accessing efficiency and better overall database manipulation performance, especially when dealing with large and complex databases.

2. A user may wish to organize and manipulate the database as a hierarchical tree structure, wherein the logical interconnections between data groupings are always one-to-many in nature. In a

sense, the manipulation of a hierarchical tree structure is a special case of the general navigational approach. Hierarchical structures do, however, allow a number of simplifications to be made in designing the database management system, as well as in the data manipulation language. Furthermore, a surprisingly large number of situations in the real world may be effectively represented with a hierarchical tree data organization, so it is worthwhile to treat hierarchical structure as an important special case.

3. Finally, in many cases it is appropriate for the applications programmer to access the database directly in terms of its underlying binary or n-ary relations (Codd, 197Ø; Codd, 1974). Such "direct" manipulation may be made at a relatively low level, in terms of individual relations and primitive operations (using the relational algebra) upon them. Alternately, a higher-level interface could be used to translate more general-purpose commands (using the relational calculus) into lower-level operations. Such low-level accessing methods generally provide greater efficiency, at the expense of greater programming detail.

2.2.3 INFOPLEX's Approach to Functional Decomposition

The above discussions illustrate one possible decomposition of the information management functions into hierarchical levels. Other decompositions are possible. For example, the work of (Senko, 1976; Yeh *et al*., 1977; Toh *et al*., 1977; ANSI, 1975) also decomposes the various information management functions into several levels (e.g., (1) physical data storage, (2) logical data encoding, (3) access path, (4) internal schema, and (5) external schema). A common weakness of these functional decompositions, including our example decompositon, is that although any particular decomposition may make good sense and impose a reasonable conceptual structure on the information management function, there are no commonly accepted criteria with which to evaluate any given decomposition.

A common qualitative criteria often used to decompose complex functions into sub-modules is that of modularity. A decomposition is considered to attain high modularity when each individual module is internally coherent, and all the modules are loosely coupled with one another. One of the INFOPLEX research focuses is to develop methodologies to formalize this notion of modularity quantitatively, and to use it to evaluate a given decomposition, thus systematic techniques for obtaining an optimal functional decomposition of the information management functions can be developed. A particularly promising approach for this purpose is the Systematic Design Methodology (Huff and Madnick, 1978). The following briefly describes this approach.

The Systematic Design Methodology approach to system design centers on the problem of identifying a system's modules, or "sub-problems", their functions, and their interconnections. Using this approach, we begin with a set of functional requirement statements for the INFOPLEX information management functions. Each pair of requirements is examined in turn, and a decision as to whether a significant degree of interdependence between the two requirements exists is made. Then the resulting information is represented as a non-directed graph structure: nodes are requirement statements, links are assessed interdependencies. The graph is then partitioned with the objective of locating a good decomposition. An index of partition goodness is employed, which incorporates measures of subgraph "strength" and "coupling". The actual goodness index is taken as the algebraic difference between the strengths of all the subgraphs, and the inter-subgraph couplings. That is, M=S-C, where S is the sum of the strength measures of all subgraphs, and C is the sum of all the inter-subgraph couplings.

Once an agreeable partition is determined, the resulting sets of requirements are interpreted as "design sub-problems". From these design sub-problems a functional hierarchy of INFOPLEX can then be systematically derived. This procedure is illustrated in Figure 2.7. This approach is currently being developed in the INFOPLEX Project.

2.3 THE INFOPLEX DATA STORAGE HIERARCHY

To provide a high performance, highly reliable, and large capacity storage system, INFOPLEX makes use of an automatically managed memory hierarchy (referred to as the INFOPLEX physical decomposition). In this section, the rationale for and an example of an automatic memory hierarchy are discussed. Then the INFOPLEX approach to realize such a memory hierarchy is also discussed.

2.3.1 Rationale for a Storage Hierarchy

The technologies that lend themselves to low cost-per-byte storage devices (and, thereby, economical large capacity storage) result in relatively slow access times. If it was possible to produce ultra-fast limitless-capacity storage devices for miniscule cost, there would be little need for a physical decomposition of the storage. Lacking such a wondrous device, the requirements of high performance at low cost are best satisfied by a mixture of technologies combin-

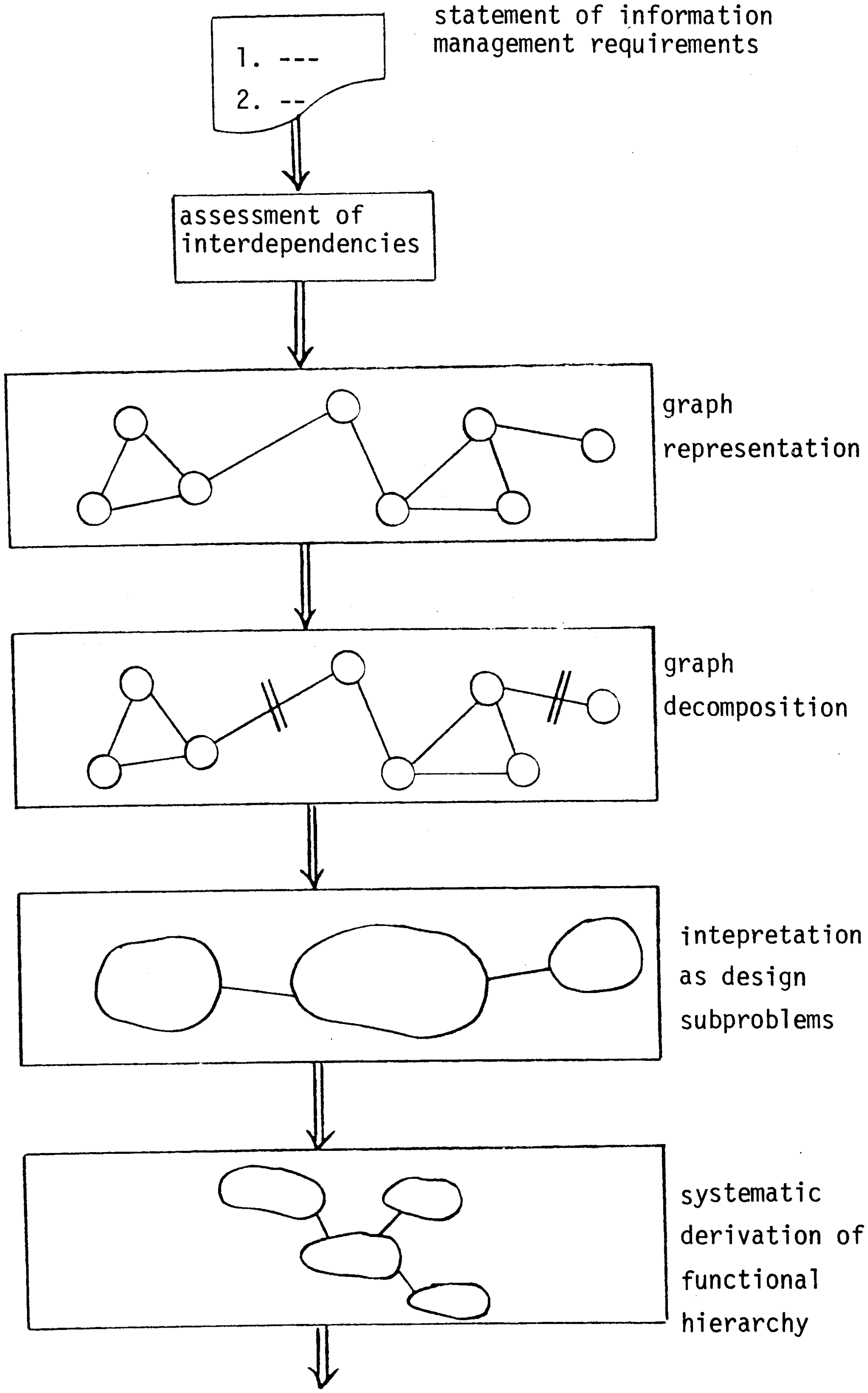

1. ---
2. --
statement of information management requirements
assessment of interdependencies
graph representation
graph decomposition
intepretation as design subproblems
systematic derivation of functional hierarchy

Figure 2.7 INFOPLEX's Approach to Functional Decomposition

ing expensive high-performance devices with inexpensive lower-performance devices.

There are many ways that such an ensemble of storage devices may be structured, but the technique of <u>hierarchical physical decomposition</u> has been found to be very effective (Madnick, 1973; Madnick, 1975a; Madnick, 1975b). Using this technique, the ensemble of heterogeneous storage devices is organized as a hierarchy. Information is moved between storage levels automatically depending upon actual or anticipated usage such that the information most likely to be referenced in the future is kept at the highest (most easily accessed) levels.

The effectiveness of a memory hierarchy depends heavily on the phenomonon known as <u>locality of reference</u> (Denning, 1970). A memory hierarchy makes use of this property of information reference pattern so that the information that is used frequently would be accessible through the higher levels of the hierarchy, giving the memory hierarchy an expected access time close to that of the access time of the faster memories. This approach has been used in contemporary computer systems in cache memory systems (Conti, 1969), in virtual memory demand paging systems (Bensoussan <u>et al</u>., 1969; Greenberg and Webber, 1975; Hatfield, 1972; Mattson <u>et al</u>., 1970; Meade, 1970), and in mass storage systems (Considine and Weis, 1969; Johnson, 1975).

Experimentations with physical data reference strings are reported in (Easton, 1978; Rodriguez-Rosell, 1976; Smith, 1978b). It has been found that there is considerable sequentiality of physical database reference in these studies. Sequentiality of references is a special form of spatial locality as discussed by (Madnick, 1973). Several measures of logical database locality and experimentations with these measures are reported in (McCabe, 1978; Robidoux, 1979). The observations from these experiments are encouraging. In particular they indicate that there is considerable locality of database reference.

2.3.2 Example of a Physical Decomposition

We now discuss an example of a memory hierarchy, its general structure, types of storage devices that it may employ, and some strategies for automatic information movement in the hierarchy.

2.3.2.1 General Structure

To the user (i.e. the lowest level of the functional hierarchy) of the memory hierarchy, the memory appears as a very large linear virtual address space with a small access time. The fact that the memory is actually a hierarchy or that a certain block of information can be obtained from a certain level is hidden from the memory user. Figure 2.8

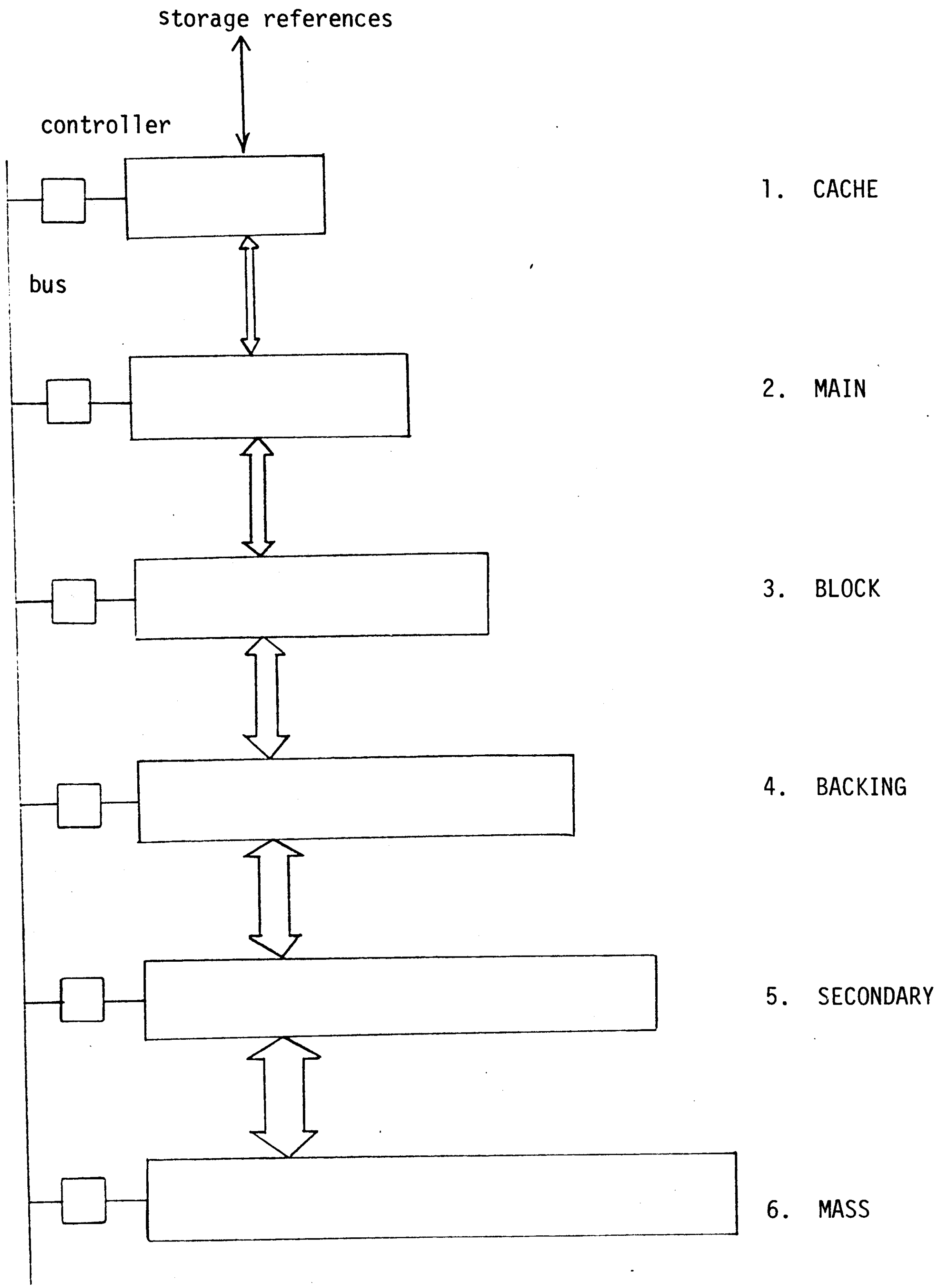


Figure 2.8 An Example Memory Hierarchy

	Storage Level	Random Access Time	Sequential Transfer Rate (bytes/sec)	Unit Capacity (bytes)	System Price (per byte)
1.	Cache	100 ns	100M	32K	50¢
2.	Main	1 us	16M	512K	10¢
3.	Block	100 us	8M	2M	2¢
4.	Backing	2 ms	2M	10M	0.5¢
5.	Secondary	25 ms	1M	100M	0.02¢
6.	Mass	1 sec.	1M	100B	0.0005¢

Figure 2.9 Example Storage Devices

illustrates the general structure of a memory hierarchy consisting of six levels of storage devices. Some of the devices that can be used in these levels are discussed in the next subsection.

The lowest level always contains all the information of the system. A high level always contains a subset of the information in the next lower level. To satisfy a request, the information in the highest (most easily accessed) level is used.

Storage reference is accomplished by supplying the memory hierarchy with a virtual address (say a 64-bit address), the memory hierarchy will determine where the addressed information is physically located. The addressed information will be moved up the memory hierarchy if it is found in other than the highest level of the hierarchy. This implies that there is a high variance in the access time of the memory system. This situation is alleviated by providing multiple ports to the memory system so that a pipeline of requests can be processed. Furthermore, the inherent parallelism within each memory level and among different memory levels provides high throughput for the memory system as a whole. Since the functional levels are designed with high parallelism of operation as one of its major objectives, the processor making the request can take advantage of the high memory access time variance. Various schemes are used to make the automatic management of the memory hierarchy efficient. Some of these strategies are discussed in a latter section.

2.3.2.2 Storage Devices

Traditionally, computer direct access storage has been dominated by two fairly distinct technologies: (1) ferrite core and, later, metallic oxide semiconductor (MOS) LSI memories with microsecond access times and relatively high costs, and (2) rotating magnetic media (magnetic drums and disks) with access time in the range of 10 to 100 milliseconds and relatively low costs. This has led to the separation between main storage and secondary storage devices.

Recently several new memory technologies, most notably magnetic bubbles, electron beam addressed memories (EBAM), and charge coupled devices (CCD), have emerged to fill the "gap" between the two traditional memory technologies.

The evolution and increasing deployment of the above and many other memory technologies have produced a more continuous cost-performance range of storage devices, as depicted in Figure 2.9 (Madnick, 1975a). Note that these technologies, which are arbitrarily grouped into six categories, result in storage devices that span more than six orders of magnitude in both random access time (from less than 100 nanoseconds to more than 1 second) and system price per byte (from more than 50 cents per byte to less than 0.0005 cent).

This evolution has facilitated the choice of appropriate cost-effective storage devices for the memory hierarchy. For example, for the memory hierarchy discussed in the previous section, we might use a device like the IBM 3850 Mass Storage as the mass storage, traditional moving head disks as secondary storage, magnetic drums as backing store, CCD or magnetic bubble as block store, core or semiconductor RAM as main storage, and high performance semiconductor RAM as cache.

2.3.2.3 Strategies for Information Movement

Various physical storage management and movement techniques, such as page splitting, read through, and store behind, can be distributed within the memory hierarchy. This facilitates parallel and asynchronous operation in the hierarchy. Furthermore, these approaches can lead to greatly increased reliability of operation. For example, under the *read through* strategy (Figure 2.10), when data currently stored at level i (and all lower performance levels) is referenced, it is automatically and simultaneously copied and stored into all higher performance levels. The data itself is moved between levels in standard *transfer*

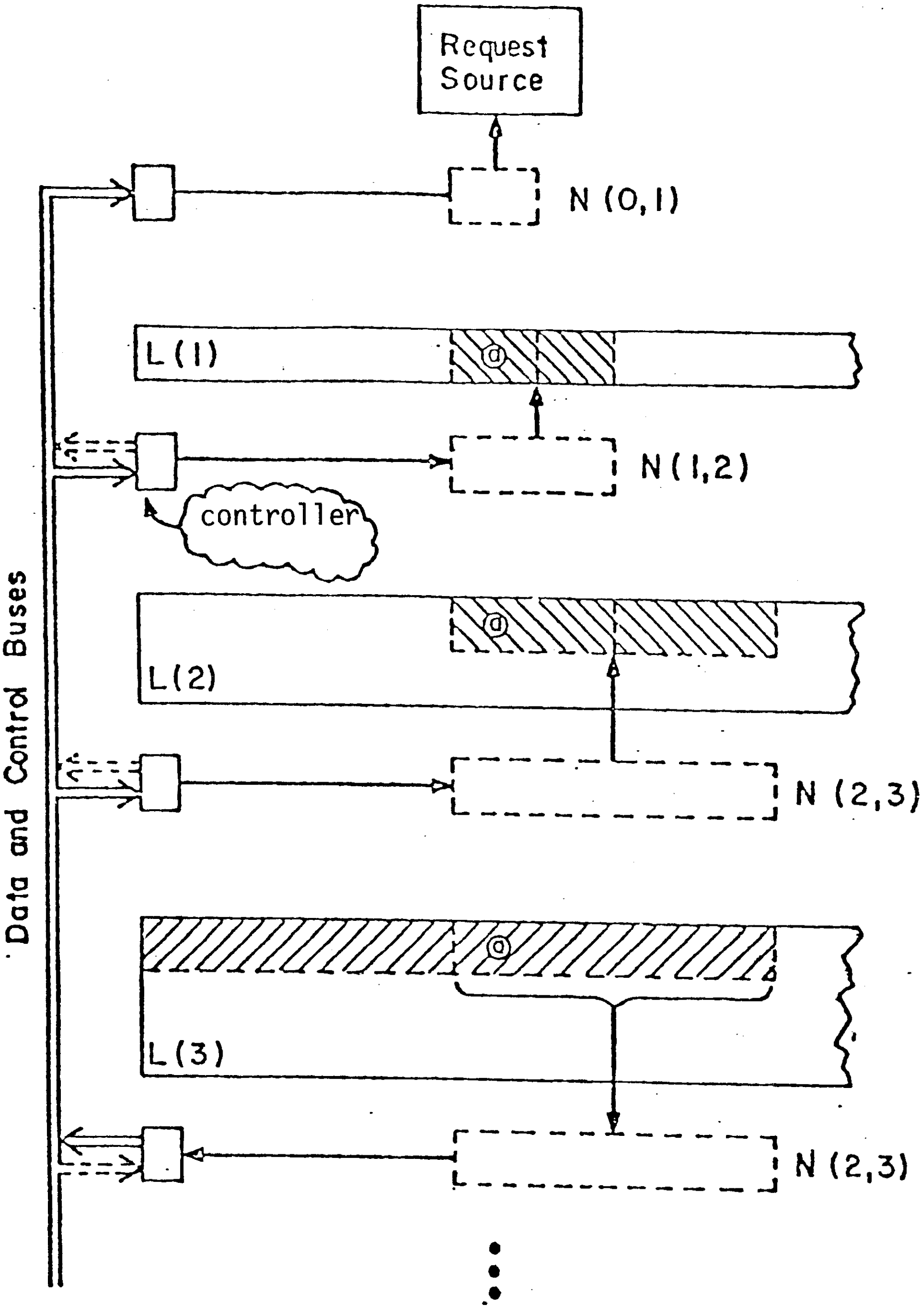


Figure 2.10 Illustration of Read Through

units, also called *pages*, whose size N (i-1, i) depends upon the storage level from which it is being moved.

For example, suppose that the datum "a", at level 3, is referenced (see Figure 2.10). The block of size N(2,3) containing "a" is extracted and moved up the data bus. Level 2 extracts this block of data and stores it in its memory modules. At the same time, level 1 extracts a sub-block of size N(1,2) containing "a" and level 0 extracts the sub-block of size N(0,1) containing "a" from the data bus.

Hence, under the read through strategy, all upper storage levels receive this information simultaneously. If a storage level must be removed from the system, there are no changes needed. In this case, the information is "read through" the level as if it didn't exist. Since all data available at level i is also available at level i + 1 (and all other lower performance levels), there is no information lost. Thus, no changes are needed to any of the other storage levels or the storage management algorithms although we would expect the performance to decrease as a result of the missing storage level. A limited form of this reliability strategy is employed in most current-day cache memory systems (Conti, 1969).

In a store behind strategy all information to be changed is first stored in L(1), the highest performance storage level. This information is marked "changed" and is copied into L(2) as soon as possible, usually during a time when there is little or no activity between L(1) and L(2). At a later time, the information is copied from L(2) to L(3), etc. A variation on this strategy is used in the MULTICS Multilevel Paging Hierarchy (Greenberg and Webber, 1975). This strategy facilitates more even usage of the bus between levels by only scheduling data transfers between levels during idle bus cycles. Furthermore, the time required for a write is only limited by the speed of the highest level memory.

The store behind strategy can be used to provide high reliability in the storage system. Ordinarily, a changed page is not allowed to be purged from a storage level until the next lower level has been updated. This can be extended to require two levels of acknowledgment. Under such a strategy, a changed page cannot be removed from L(1) until the corresponding pages in both L(2) and L(3) have been updated. In this way, there will be at least two copies of each changed piece of information at levels L(i) and L(i+1) in the hierarchy. Other than slightly delaying the time at which a page may be purged from a level, this approach does not significantly affect system performance. As a result of this technique, if any level malfunctions, it can be removed from the hierarchy without causing any information to be lost. There are two exceptions to this process, L(1) and L(n). To completely safeguard the reliability of the system, it may be necessary to store duplicate copies of information at these levels only.

Figure 2.11 illustrates this process. In Figure 2.11(a), a processor stores into L(1), the corresponding page is marked "changed" and "no lower level copy exists". Figure 2.11(b) shows in a latter time, the corresponding page in L(2) is updated and marked "changed" and "no lower level copy exists". An acknowledgment is sent to L(1) so that the corresponding page is marked "one lower level copy exists". At a later time (Figure 2.11(c)), the corresponding page in L(3) is updated and marked "changed" and "no lower level copy exists". An acknowledgment is sent to L(2) so that the corresponding page is marked "one lower level copy exists". An acknowledgment is sent to L(1) so that the corresponding page is marked "two lower level copy exists". At this time, the page in L(1) may be replaced if necessary, since then there will be at least two copies of the updated information in the lower memory levels.

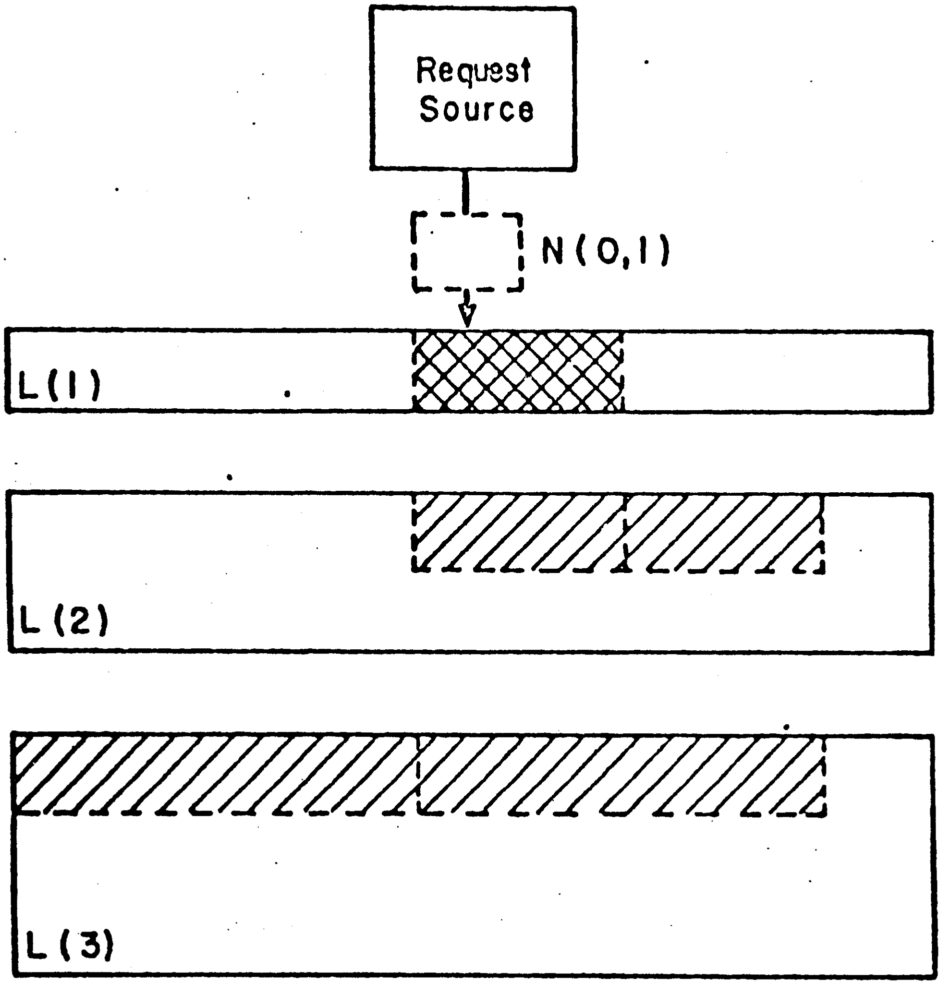


Figure 2.11(a) Store Behind (a)

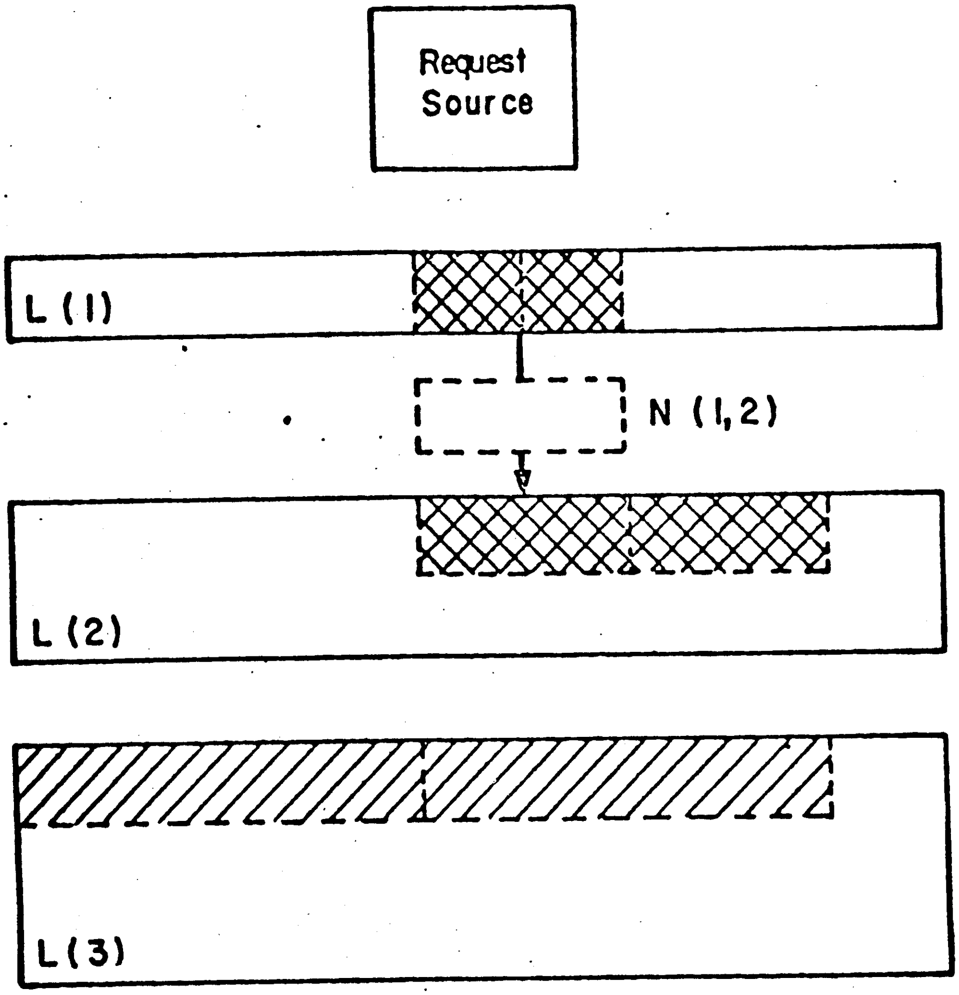


Figure 2.11(b) Store Behind (b)

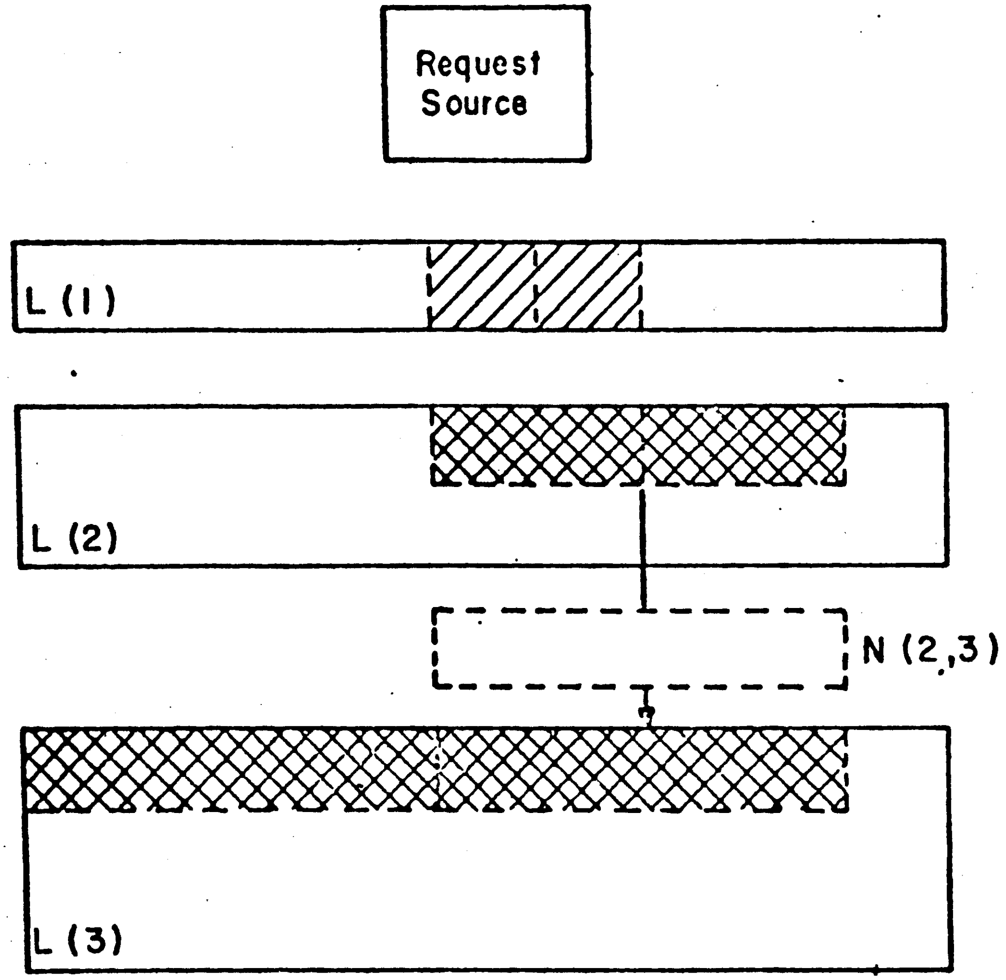


Figure 2.11(c) Store Behind (c)

2.3.3 INFOPLEX's Approach to Physical Decomposition

In the previous section, we have illustrated an example of a memory hierarchy that makes use of an ensemble of heterogeneous storage devices. Although memory hierarchies using two or three levels of storage devices have been implemented, no known generalized automatic memory hierarchy has been developed.

The optimality of a memory hierarchy depends on the complex interactions among the memory reference pattern, the device characteristics, and the information movement strategies. The INFOPLEX approach to this complex problem is to empirically study and characterize data reference patterns at several levels (e.g. transaction level, logical data level, and physical data level), to develop various information movement strategies, and to design a prototype memory hierarchy. The interactions among these components can then be systematically investigated by means of analytic models and simulation models.

2.4 RESEARCH ISSUES ADDRESSED IN THIS THESIS

This chapter has provided the background for this thesis. As is evident from the above discussions, there are a large number of interesting but unresolved research problems associated with INFOPLEX. This thesis is a key step towards understanding the INFOPLEX data storage hierarchy. In particular, this thesis has made contributions in the following areas:

1. Developed and extended concepts for the INFOPLEX data base computer and data storage hierarchy systems.
2. Provided a theoretic foundation for analysis of data storage hierarchy systems.
3. Formalized storage management algorithms to incorporate the read-through strategy.
4. Provided detail analysis of the performance and reliability properties of data storage hierarchies and their algorithms.
5. Designed prototype data storage hierarchy systems for INFOPLEX.
6. Developed simulation models to obtain insights to the data storage hierarchy designs and their algorithms.

These are elaborated in the following chapters.

Chapter III

A GENERAL STRUCTURE OF THE INFOPLEX DATA STORAGE HIERARCHY

3.1 INTRODUCTION

This chapter proposes the general structure of a data storage hierarchy system for the INFOPLEX data base computer. The design of this system is based on Madnick's proposed system (Madnick, 1973). This work brings Madnick's system one step closer to realization. In the following, the underlying design goals of this data storage hierarchy system will be discussed. Then the design, called DSH-1, is introduced followed by a discussion of further design issues that need to be addressed.

3.2 DESIGN OBJECTIVES

There are a large number of practical storage hierarchy systems today. However, the functionality provided by each is quite different and often falls short of our expectations (for use as the storage subsystem of the INFOPLEX data base computer). In the following, we discuss the underlying design goals of DSH-1.

3.2.1 Virtual Address Space

DSH-1 provides a virtual address space for data storage. Every data item in DSH-1 is byte addressable using a generalized virtual address. A key advantage of a virtual address space is that a user (a processor) of DSH-1 is relieved of all physical device concerns. In fact, the processor accessing DSH-1 is not aware of how the virtual address space is implemented. This latter characteristic is quite unique since most current virtual memory systems are simulated, at least partially, by software executed by the processor, e.g., the IBM OS/VS system (Scherr, 1973).

3.2.2 Very Large Address Space

Early virtual memory systems were developed primarily for program storage, hence their address spaces were quite limited, e.g., in the order of one million bytes. The MULTICS (Greenberg and Webber, 1975) virtual memory and the IBM System/38 (Datamation, 1978; Soltis and Hoffman, 1979) logical storage were developed for program as well as data file storage. These systems support a large virtual address space. However, the size of an individual data file in MULTICS is limited to 2**18 bytes and that in System/38 is limited to 2**24 bytes. Though these are very large address spaces, it is expected that future systems will require online storage capacities that are much larger. DSH-1 uses a 64-bit virtual address. Each byte is directly addressable, hence there is virtually no limit on the size of a logical entity such as a data file.

3.2.3 Permanent Data Storage

Accesses to permanent data is performed by special software routines and a special I/O processor in most virtual memory systems. The I/O processor brings the data into the virtual memory and writes the data back to permanent storage when the data is updated. Systems like MULTICS and System/38 provide a permanent virtual data storage. Any data in virtual memory is also in permanent storage. DSH-1 also provides a permanent virtual data storage. Special data integrity schemes are used to ensure that as soon as a processor completes a write operation to a virtual location, the effect of the write becomes permanent even in the event of a power failure.

3.2.4 Support Multiple Processors

Most current virtual memory systems have been limited to supporting 2 to 3 processors. It is necessary that DSH-1 support a large number of processors due to the requirements for high performance and high availability to be discussed below. All these processors share the same virtual data address space. Appropriate synchronization and protection schemes are used to ensure data integrity and security.

3.2.5 Generalized Multi-level Storage System

To provide a large capacity storage subsystem with low cost and high performance, a spectrum of storage devices arranged in a hierarchy is used. Previous storage hierarchy systems have been specially designed for a specific 2 or 3 levels hierarchy (e.g., cache and main memory, or main memory and secondary storage device). Thus, it is extremely difficult to add or remove a storage level in these systems. DSH-1 is designed to incorporate any type of storage device and support reconfiguration of storage levels. This characteristic is particularly important in responding to new device technologies.

3.2.6 Direct Inter-level Data Transfer

In most current storage hierarchy systems, data movement among storage levels is performed indirectly. For example, to move data from drum to disk in the MULTICS system, data is read from drum into main memory by the processor which then writes the data to disk. Recent developments in storage systems make it possible to decentralize the control of data movement between storage devices to intelligent controllers at the storage devices. For example, the IBM 3850 Mass Storage (Johnson, 1975) uses an intelligent controller to handle data transfer between mass storage and disks, making the 3850 appear as a very large number of virtual disks.

DSH-1 incorporates intelligent controllers at each storage level to implement the algorithms for data movement among the storage levels. Special algorithms are developed to facilitate efficient broadcasting of data from a storage level to all other storage levels as well as movement of data between adjacent storage levels.

3.2.7 High performance

To support the data requirements of the functional processors in INFOPLEX, DSH-1 is designed to handle a large number of requests simultaneously. The operation of DSH-1 is highly parallel and asychronous. Thus, many requests may be in different stages of completion at various storage levels of DSH-1. Each processor accesses DSH-1 through a data cache where the most frequently used data items are stored.

3.2.8 Availability

High availability of DSH-1 is a result of a combination of the design strategy used, hardware commonality, and special algorithms. Key design strategies in DSH-1 include the use of distributed controls and simple bus structures, both of which contribute to the high availability of DSH-1. Multiple identical hardware components are used in parallel to provide high performance and to ensure that no single component is critical to system operation. Integrated into the design are certain algorithms that exploit the structure of DSH-1 to allow data redundancy and perform automatic data repair in the event of component failure, thus diminishing the dangers of multiple failures.

3.2.9 Modularity

DSH-1 is modular at several levels. This provides much flexibility in system structuring. The number of processors to be supported by DSH-1 can be varied. The number of storage levels and the type of storage devices can be chosen to meet the particular capacity and performance requirements. All the storage levels have very similar structures and the same algorithm is used by the intelligent controllers at each storage level.

Flexibility in system structuring is extended in DSH-1 to allow for dynamic system reconfiguration. For example, a defective storage device or storage level can be amputated without loss of system availability.

An example of a system that also incorporates modularity as a key design goal is the PLURIBUS (Katsuki _et_. _al_., 1978) system. In PLURIBUS, the basic building block is a _bus module_. The number of components on a bus module as well as the number of bus modules can be easily varied to meet different system requirements.

3.2.1Ø Low Cost

A storage hierarchy is the lowest cost configuration to meet the requirement of providing a large storage capacity with high performance. DSH-1 also make use of common hardware modules as the intelligent controllers at each storage level, thus reducing hardware development cost. The modularity features of DSH-1 discussed above also facilitate system upgrading with minimum cost.

Commonality of hardware modules and flexibility of system upgrade have been employed in many computer systems as an effective approach to reduce cost. However, these techniques are rarely applied to storage hierarchy systems. DSH-1 is a step in this direction.

Advances in storage device and processor technologies provide great potentials for development of very effective data storage hierarchies that incorporate the above characteristics. In the next section, we describe a general structure of such a system.

3.3 GENERAL STRUCTURE OF DSH-1

The structure of DSH-1 is illustrated in Figure 3.1. A key design decision in DSH-1 is to make use of an asynchronous time-shared bus for interconnecting multiple components (processors and memory modules) within a storage level and

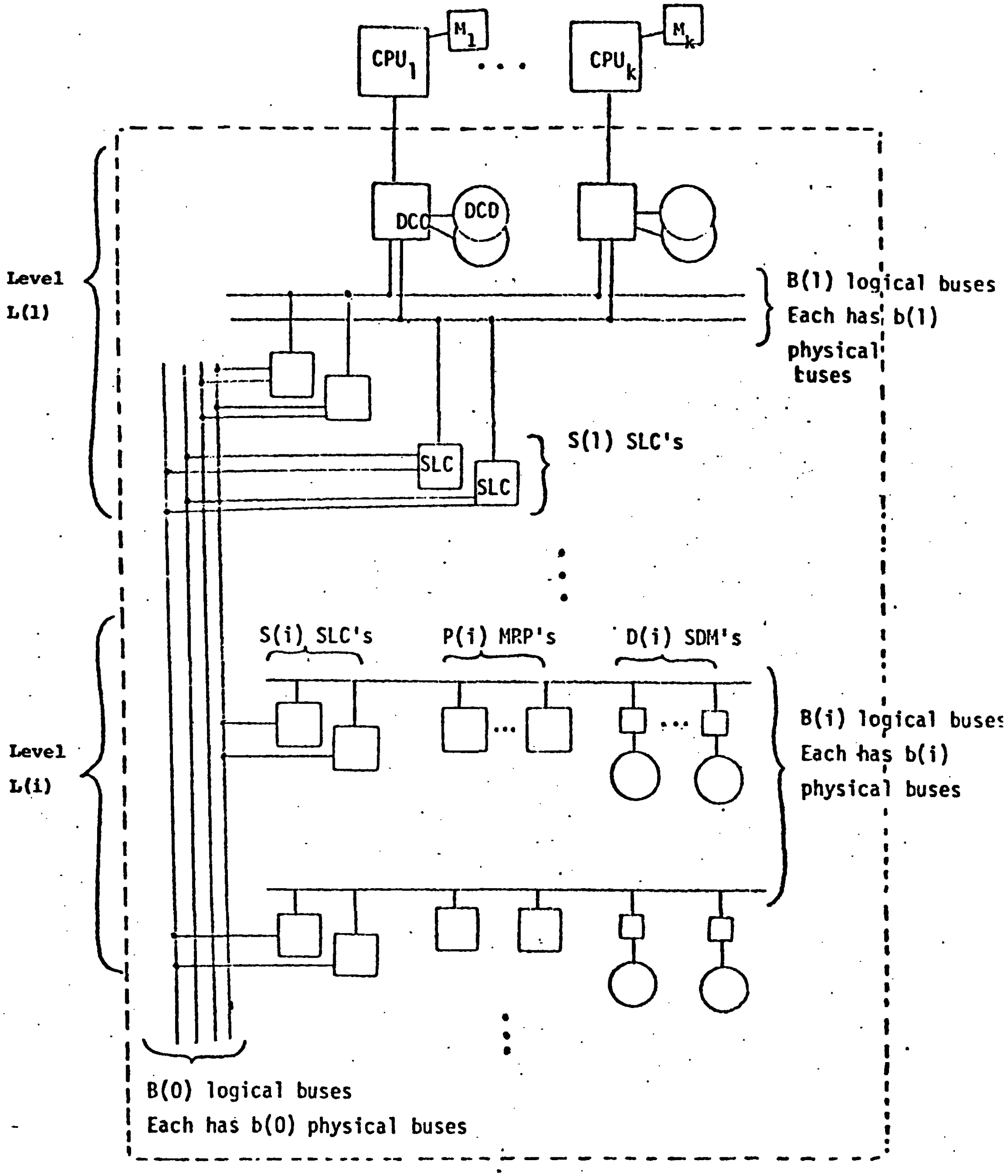

CPU1
M1
CPUk
Mk
DCC
DCD
Level L(1)
B(1) logical buses Each has b(1) physical buses
SLC
SLC
S(1) SLC's
S(i) SLC's
P(i) MRP's
D(i) SDM's
Level L(i)
B(i) logical buses Each has b(i) physical buses
B(0) logical buses
Each has b(0) physical buses

Figure 3.1 Structure of DSH-1

to make use of an asynchronous time-shared bus for interconnecting all the storage levels. A key advantage of the time-shared bus is its simplicity, flexibility, and throughput. Two alternative approaches can be used in DSH-1 to increase the effective bandwidth of the time-shared bus. First, a new pended-bus protocol can be used (Haagens, 1978). This asynchronous bus protocol is more efficient than the usual time-shared bus protocols with the result that a much larger number of components can share a single bus. Second, multiple logical buses can be used to partition the load on the time-shared bus.

In the following subsections, we shall describe the interface to DSH-1 as seen by a functional hierarchy processor. Then the structure of DSH-1 is described by examining its highest performance storage level and then a typical storage level.

3.3.1 The DSH-1 Interface

To the functional hierarchy processors connected to DSH-1, DSH-1 appears as a large multi-port main memory. There are K memory ports, hence K processors can simultaneously access DSH-1.

The functional processors use a 2**V (V=64) byte virtual address space. The instructions for each functional hierarchy processor are stored in a separate 2**I byte program memory. The program memories are not part of DSH-1. Thus, 2**I bytes of the processor's address space is mapped by the program memories, leaving 2**V-2**I bytes of data memory to be managed by DSH-1. This is depicted in Figures 3.2(a) and 3.2(b).

Each processor has multiple register sets to support efficient multiprogramming. Some of the more important registers for interfacing with DSH-1 are : (1) a V-bit Memory Address Register (MAR) for holding the virtual address, (2) a Memory Buffer Register (MBR) for storing the data read from DSH-1 and to be written into DSH-1, (3) a Memory Operation Register (MOR) indicates the particular operation to be performed by DSH-1, (4) an Operation Status Register (OSR) which indicates the result of a operation performed by DSH-1, and (5) a Process Identifier Register (PIR) which contains the Process Identifier (PID) of the process that is currently using the processor.

A number of memory operations are possible. The key ones are the read and write operations and the primitives for locking a data item (such as those supporting the Test-and-Set type of operations).

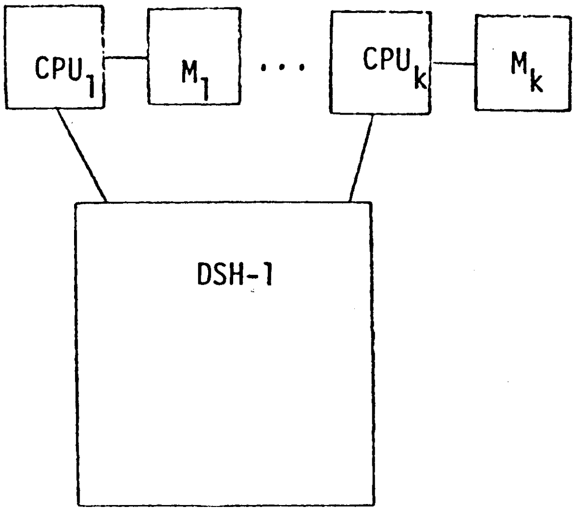


Figure 3.2(a) The DSH-1 Interface

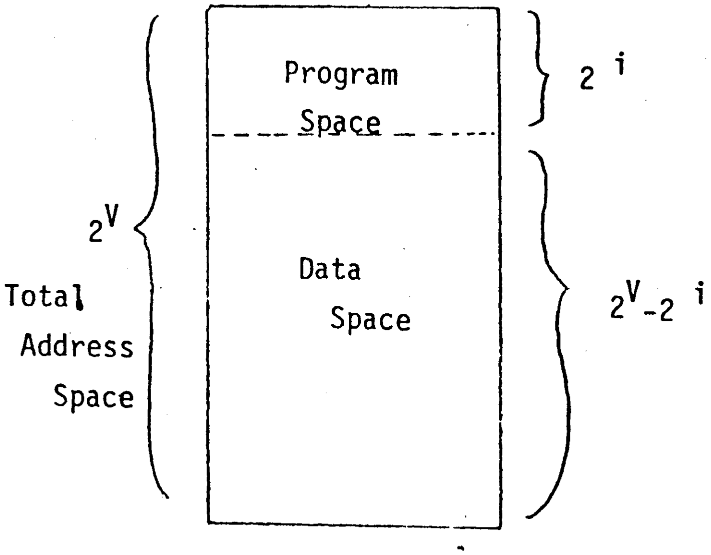


Figure 3.2(b) The DSH-1 Address Space

All read and write operations to DSH-1 are performed in the highest performance storage level, L(1). If a referenced data item is not in L(1), it is brought up to L(1) from a lower storage level via a read-through operation. The effect of an update to a data item in L(1) is propagated down to the lower storage levels via a number of store-behind operations.

In a read operation, two results can occur depending on the state of DSH-1. First, if the requested data is already in L(1), the MBR is filled with the data bytes starting at location (MAR) and the processor continues with the next operation. Alternatively, the addressed data may not be available in L(1). In this case, the processor is interrupted, the OSR is set to indicate that it may take a while for the read operation to complete, and the processor is switched to another process. In the meantime, the addressed data is copied into L(1) from a lower storage level. When this is completed, the processor is notified of the completion of the original read operation.

Similarly, a write operation may result in two possible responses from DSH-1. First, if the data to be updated is already in L(1), the bytes in MBR are written to the virtual address locations starting at (MAR), and the processor continues with the next operation. Second, a delay similar to the read operation may occur (when the data to be updated is not in L(1)), while DSH-1 retrieves the data from a lower storage level.

This concludes a brief description of the asynchronous DSH-1 interface, as seen by a functional hierarchy processor. Next, we examine the structure of DSH-1.

3.3.2 The Highest Performance Storage Level - L(1)

There are h storage levels in DSH-1, labelled L(1), L(2), L(3), ..., L(h). L(1) is the highest performance storage level. L(i) denotes a typical storage level. The structure of L(1) is unique. The structures of all other storage levels are similar.

A distinction must be made between the concept of a physical bus and a logical bus. The former refers to the actual hardware that implements communications among levels and within a level. A logical bus may be implemented using one or more physical buses. Logical buses represent a partitioning, based upon the virtual address referenced, of the physical buses.

Referring to Figure 3.1, L(1) consists of K memory ports and S(1) storage level controllers (SLC's) on each of B(1) logical local buses (i.e., S(1)*B(1) SLC's in total for this level). Each memory port consists of a data cache controller (DCC) and a data cache duplex (DCD). A DCC interfaces with the functional hierarchy processor that is connected to the memory port. A DCC also performs mapping of a virtual address generated by the processor to a physical address in the DCD. Another function of DCC is to interface with other DCC's (e.g., to maintain data cache consistency), and with SLC's on the logical bus (for communications with other storage levels).

At L(1), a SLC accepts requests to lower storage levels from the DCC's and forwards them to a SLC at the next lower storage level. When the responses to these requests are ready, the SLC accepts them and sends them back to the appropriate DCC's. The SLC's also couple the local buses to the global buses. In essence, the SLC serves as a gateway between levels and they contend among themselves for use of the communication media, the logical buses.

At L(1), there are B(1) logical local buses. Each logical local bus consists of b(1) physical buses. Each logical bus handles a partition of the addresses. For example, if two logical buses were used, one might handle all odd numbered data blocks and the other would handle all the even numbered data blocks.

DSH-1 has B(Ø) logical global buses. Each logical global bus consists of b(Ø) global physical buses. The use of address partitioning increases the effective bus bandwidth. The use of multiple physical buses for each logical bus enhances reliability and performance.

3.3.3 A Typical Storage Level - L(i)

A typical storage level, L(i), is divided into B(i) address partitions. Each address partition consists of S(i) SLC's, P(i) memory request processors (MRP's), and D(i) storage device modules (SDM's), all sharing a logical bus. A logical bus consists of b(i) physical buses.

An SLC is the communication gateway between the MRP's/SDM's of its level and the other storage levels.

An MRP performs the address mapping function. It contains a directory of all the data maintained in the address partition. Using this directory, an MRP can quickly determine if a virtual address corresponds to any data in the address partition, and if so, what the real address is for the data. This real address can be used by the corresponding SDM to retrieve the data. Since each MRP contains a copy of this directory, updates to the directory have to be handled with care, so that all the MRP's see a consistent copy of the directory.

An SDM performs the actual reading and writing of data. It also communicates with the MRP's and the SLC's.

The SLC's, MRP's, and SDM's cooperate to handle a memory request. An SLC communicates with other storage levels and passes requests to an MRP to perform the address translation. The appropriate SDM is then initiated to read or write the data. The response is then sent to another SLC at another storage level.

3.4 FURTHER DESIGN ISSUES

The previous section describes the general structure of DSH-1. From this general structure, a number of interesting alternative configurations can be obtained. For example, if all the data caches are taken away, L(1) becomes a level with only the SLC's for communicating the requests from the processors to the lower storage levels and for obtaining responses from these lower storage levels. This configuration eliminates the data consistency problems associated with multiple data caches.

If we let the number of logical buses be equal to one, we obtain the configuration without address partitioning.

Another intersting configuration is when there is only one MRP and one SDM on a given logical bus. This configuration eliminates the need for multiple identical directory updates.

Thus, by varying the design parameters of DSH-1, a large number of alternative configurations with quite different characteristics can be obtained. The general structure is a valuable vehicle for investigating various design issues. Some of the key issues are briefly introduced in the following sections.

3.4.1 Support of Read and Write Operations

Key problems in supporting the read and write operations in DSH-1 include : (1) data consistency in multiple data caches, (2) protocols for communicating over the shared bus, (3) algorithms for updating the redundant directories, (4) algorithms for arbitrating among usage of identical resources, such as buses, SLC's and MRP's, and (5) specifying the various steps (transactions) that have to be accomplished to handle the read and write operations.

3.4.1.1 Multiple Cache Consistency

As illustrated in Figure 3.1, each DSH-1 memory port is a data cache directly addressable by the processor at the port. It is possible then, that a data item may be in several different data caches at the same time. When the data item gets updated by a processor, other processors may reference an inconsistent copy of the data item. The multiple cache consistency problem and its solutions are discussed in (Tang, 1976; Censier and Feautrier, 1978).

Three basic approaches can be used to resolve this problem in DSH-1. The first approach is to send a purge request to all other data caches whenever a processor updates data in its cache. The second approach is to maintain status information about the data cache contents. Whenever there is an update to a data item, this status information is consulted and purge requests are sent only to those caches that contain the data item being changed. The third approach is to make use of knowledge of how the data in DSH-1 is to be used so that the inconsistency problem can be avoided. For example, knowledge about the interlocking scheme used to ensure safe data sharing may be used to avoid uncessary purge requests to other caches.

3.4.1.2 Bus Communication Protocols

In DSH-1, the buses may be used for point-to-point communication as well as for broadcast type of communications. It is necessary to ensure that messages are sent and received correctly. For example, L(i) broadcast data to the upper levels and one or more of these levels may not be able to accomodate the data to be received, possibly due to the lack of buffer space. Communications protocols to handle these situations are important.

3.4.1.3 Multiple Directory Update

Each MRP contains a directory of all the data in the SDM's on the same bus. Multiple requests may be handled by the MRP's. When a MRP updates its directory, other MRP's may still reference the old copy of the directory. This is similar but not identical to the multiple cache consistency problem discussed above. It is necessary to maintain consistency of the MRP directory states.

3.4.1.4 Multiple Resource Arbitration

Multiple identical resources (e.g., buses, MRP's, and SLC's) are used in DSH-1 to provide parallel processing while at the same time providing redundancy against failure. A request for a resource can be satisfied by any one of the resources. An arbitration scheme is required to control the assignment of resource.

3.4.1.5 Transaction Handling

A read or a write request may go through a number of asynchronous steps through a number of storage levels to completion. A complication to these transactions is that for high throughput, a request (or response) may be divided into a number of messages when the request (or response) is being transported within the hierarchy. Thus, a request (or response) may have to be assembled, which may take an amount of time dependent on the traffic within DSH-1. Partial requests (responses) at a storage level require special handling.

3.4.2 Multiple Data Redundancy Properties

As a result of the read-through operation, several copies of a referenced data item exists in the DSH-1 storage levels. The two-level store-behind operation also maintains at least two copies of any updated data item in DSH-1. This is a key reliability feature of DSH-1. It is important to know under what conditions and using what types of algorithms can this multiple data redundancy be maintained at all times.

3.4.3 Automatic Data Repair Algorithms

One of the benefits of maintaining redundant data in DSH-1 is that lost data due to component failures can be reconstructed on a spare component from a copy of the lost data. By using automatic data repair in DSH-1 the probability of multiple data loss can be reduced.

Two classes of automatic data repair algorithms are possible. One strategy is to make use of the multiple data redundancy properties of DSH-1 and to reconstruct the lost data from its copy in a different storage level. The other approach is to maintain duplicate copies of the data item within a storage level and to reconstruct the lost data from its copy in the same storage level. The latter approach is particularly attractive for low performance devices such as mass storage.

3.4.4 Performance Evaluation

A key issue in the DSH-1 design is predicting its performance. In order to accomplish this, a simplified design of DSH-1 and its algorithms can be developed. A simulation model can then be developed for this design. Various basic performance statistics can then be obtained under various load assumptions. This experiment will provide insights and directions for further design efforts.

3.5 SUMMARY

The INFOPLEX storage hierarchy is a high performance high availability virtual memory data storage hierarchy with distributed controls for data movement and address translation. It is designed specifically to provide a very large permanent virtual address space to support multiple functional hierarchy processors.

A general structure of DSH-1, the INFOPLEX storage hierarchy has been described in this chapter. This general structure can be used to derive a large number of alternative configurations which can be used to explore various algorithms for data storage hierarchy systems.

A number of important design issues associated with DSH-1 are also outlined.

Chapter IV

MODELLING AND ANALYSIS OF DATA STORAGE HIERARCHY SYSTEMS

4.1 INTRODUCTION

This chapter is aimed at modelling data storage hierarchy systems so as to study these systems from a theoretic point of view. These studies provide insights to the perfromance and reliability properties of data storage hierarchy systems and their algorithms. These insights provide guidance to developing effective data storage hierarchy systems.

Current research in storage hierarchy systems is reviewed and extended. A formal model of a data storage hierarchy which incorporates multiple page sizes and maintains multiple data redundancy is developed. The LRU algorithm is extended to include the read-through and overflow handling strategies in a multi-level storage hierarchy. Formal definitions for these extended algorithms are developed. Finally, important performance and reliability properties of data storage hierarchy systems are identified and analyzed in detail.

4.2 RESEARCH ON STORAGE HIERARCHY SYSTEMS

Two and three-level memory hierarchies have been used in practical computer systems (Conti, 1969; Johnson, 1975; Greeberg and Webber, 1975). However, there is relatively little experience with general hierarchical storage systems.

One major area of theoretic study of storage hierarchy systems in the past has been the optimal placement of information in a storage hierarchy system. Three approaches to this problem have been used: (1) Static placement (Ramamoorthy and Chandy, 1970; Arola and Gallo, 1971; Chen, 1973) - this approach determines the optimal placement strategy statically, at the initiation of the system; (2) Dynamic placement (Lum et al, 1975; Franaszek and Bennett, 1978) - this approach attempts to optimally place information in the hierarchy, taking into account the dynamically changing nature of access to information; (3) Information structuring (Hatfield and Gerald, 1971; Jobnson J., 1975) - this approach manipulates the internal structure of information so that information items that are frequently used together are placed adjacent to each other.

Another major area of theoretic study of storage hierarchy systems has been the study of storage management algorithms (Belady, 1966; Belady et al, 1969; Denning, 1970; Mattson et al, 1970; Mattson, 1971; Hatfield, 1972; Madnick, 1973; Goldberg, 1974; Franklin et al, 1978). Here the study of storage hierarchies and the study of virtual memory systems for program storage have overlapped considerably. This is largely due the fact that most of the studies of storage hierarchies in the past have been aimed at providing a virtual memory for program storage. These studies usually do not consider the effects of multiple page sizes across storage levels, nor the problem of providing redundant data across storage levels as used in the system proposed by Madnick (Madnick, 1973). These considerations are of great importance for a storage hierarchy designed specifically for very large data bases. The following sections extend theories on storage hierarchy to include systems that incorporate multiple page sizes and maintains multiple data redundancy.

4.3 MODEL OF A DATA STORAGE HIERARCHY

A data storage hierarchy consists of h levels of storage devices, M^1, M^2, ..., M^h. The page size of M^i is Q_i and the size of M^i is m_i pages each of size Q_i. Q_i is always an integral multiple of Q_{i-1}, for i=2,3 ..., h. The unit of information transfer between M^i and M^{i+1} is a page, of size Q_i. Figure 4.1 illustrates this model of the data storage hierarchy.

All references are directed to M^1. The storage management algorithms automatically transfer information among storage levels. As a result, the data storage hierarchy appears to the reference source as a M^1 storage device with the size of M^h.

As a result of the storage management algorithms (to be discussed next), multiple copies of the same information may exist in different storage levels.

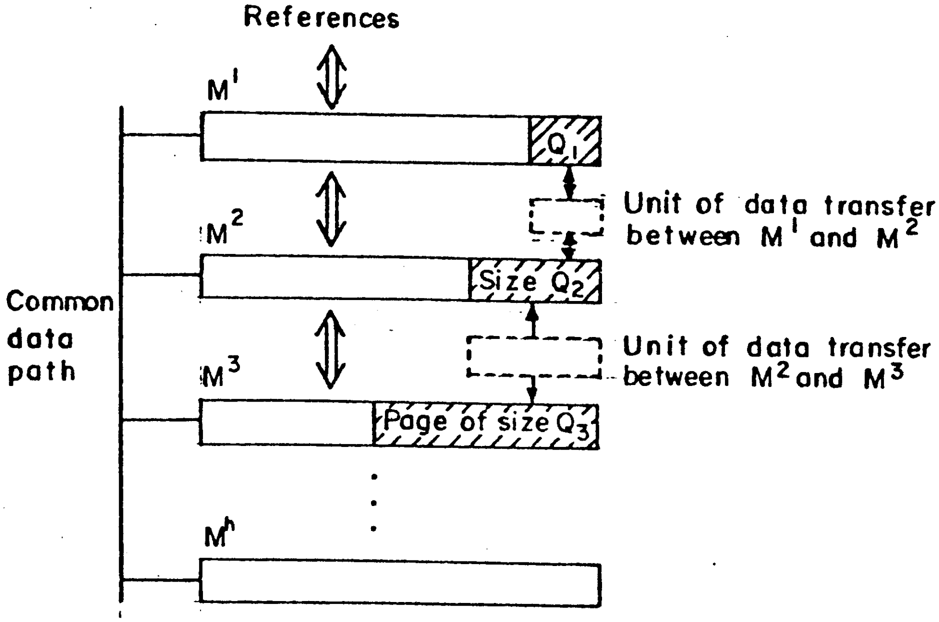

References
M^1
Q_1
Unit of data transfer between M^1 and M^2
M^2
Size Q_2
Common data path
Unit of data transfer between M^2 and M^3
M^3
Page of size Q_3
M^h

Figure 4.1 Model of a Data Storage Hierarchy

4.3.1 Storage Management Algorithms

We shall focus our attentions on the basic algorithms to support the read-through (Madnick, 1975) operation. Algorithms to support other operations can be derived from these basic algorithms.

In a read-through, the highest storage level that contains the addressed information broadcasts the information to all upper storage levels, each of which simultaneously extracts the page (of the appropriate size) that contains the information from the broadcast. If the addressed information is found in the highest storage level, the read-through reduces to a simple reference to the addressed information in that level. Figure 4.2 illustrates the read-through operation.

Note that in order to load a new page into a storage level an existing page may have to be displaced from that storage level. We refer to this phenomenon as overflow. Hence, the basic reference cycle consists of two sub-cycles, the read-through cycle (RT), and the overflow handling cycle (OH), with RT preceeding OH.

For example, Figure 4.2 illustrates the basic reference cycle to handle a reference to the page p^1_{ya}. During the Read-Through (RT) subcycle, the highest storage level (M^x) that contains p^1_{ya} broadcasts the page containing P^1_{ya} to all upper storage levels, each of which extracts the

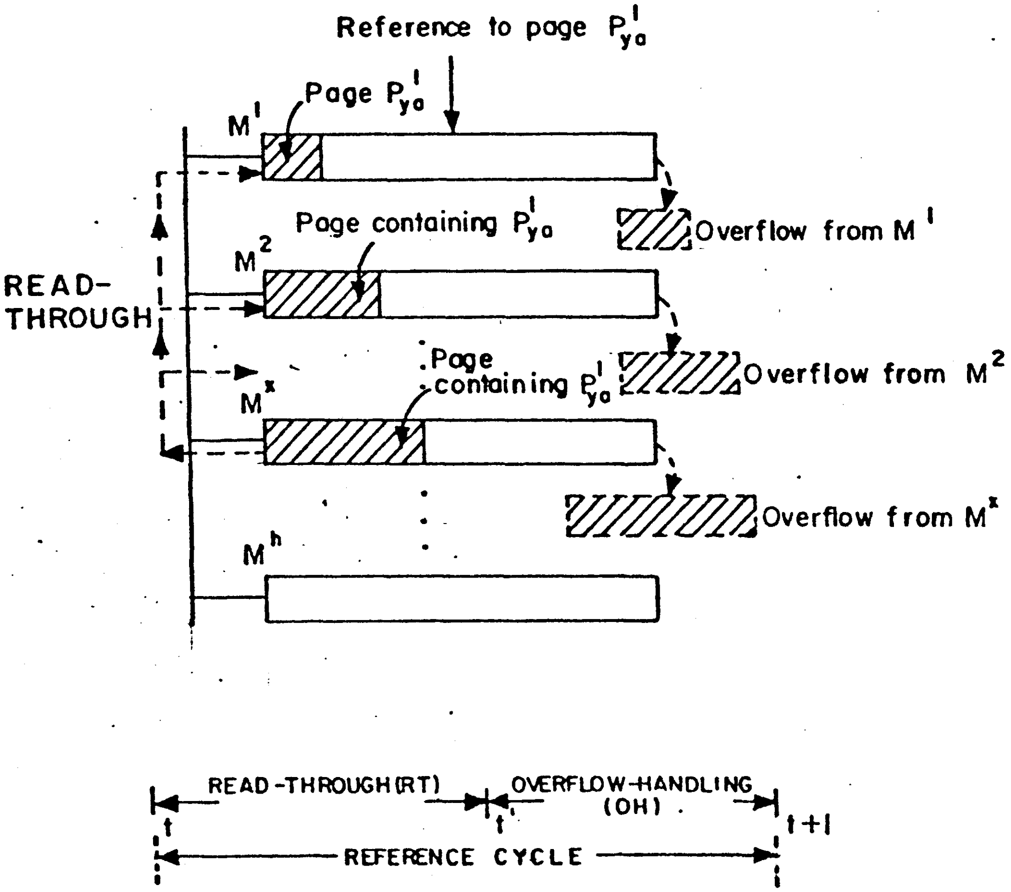


Figure 4.2 The Read Through Operation

page of appropriate size that contains P^1_{ya} from the broadcast. As result of the Read-Through, there may be overflow from the storage levels. These are handled in the Overflow-Handling (OH) subcycle.

It is necessary to consider overflow handling because it is desirable to have information overflowed from a storage level to be in the immediate lower storage level, which can then be viewed as an extension to the higher storage level.

One strategy of handling overflow to meet this objective is to treat overflows from M^i as references to M^{i+1}. We refer to algorithms that incorporate this strategy as having _dynamic-overflow-placement_ (DOP).

Another possible overflow handling strategy is to treat an overflow from M^i as a reference to M^{i+1} only when the overflow information is not already in M^{i+1}. If the overflow information is already in M^{i+1}, no overflow handling is necessary. We refer to algorithms that incorporate this strategy as having _static-overflow-placement_ (SOP).

Let us consider the algorithms at each storage level for selecting the page to be overflowed. Since the Least Recently Used (LRU) algorithm (Denning, 1974; Mattson _et_. _al_, 1970) serves as the basis for most current algorithms, we shall consider natural extensions to LRU for managing the storage levels in the data storage hierarchy system.

Consider the following two strategies for handling the Read-Through Cycle. First, let every storage level above and including the level containing the addressed information be updated according to the LRU strategy. Thus, all storage levels lower than the addressed information do not know about the reference. This class of algorithms is called LOCAL-LRU algorithm. This is illustrated in Figure 4.3.

The other class of algorithms that we shall consider is called GLOBAL-LRU algorithm. In this case, all storage levels are updated according to the LRU strategy whether or not that level actually participates in the read-through. This is illustrated in Figure 4.4.

Although the read-through operation leaves supersets of the page p_{ya}^1 in all levels, the future handling of each of these pages depends upon the replacement algorithms used and the effects of the overflow handling. We would like to guarantee that the contents of each storage level, M^i, is always a superset of its immediately higher level, M^{i-1}. This property is called Multi-Level Inclusion (MLI). Conditions to guarantee MLI will be derived in a later section.

It is not difficult to demonstrate situations where

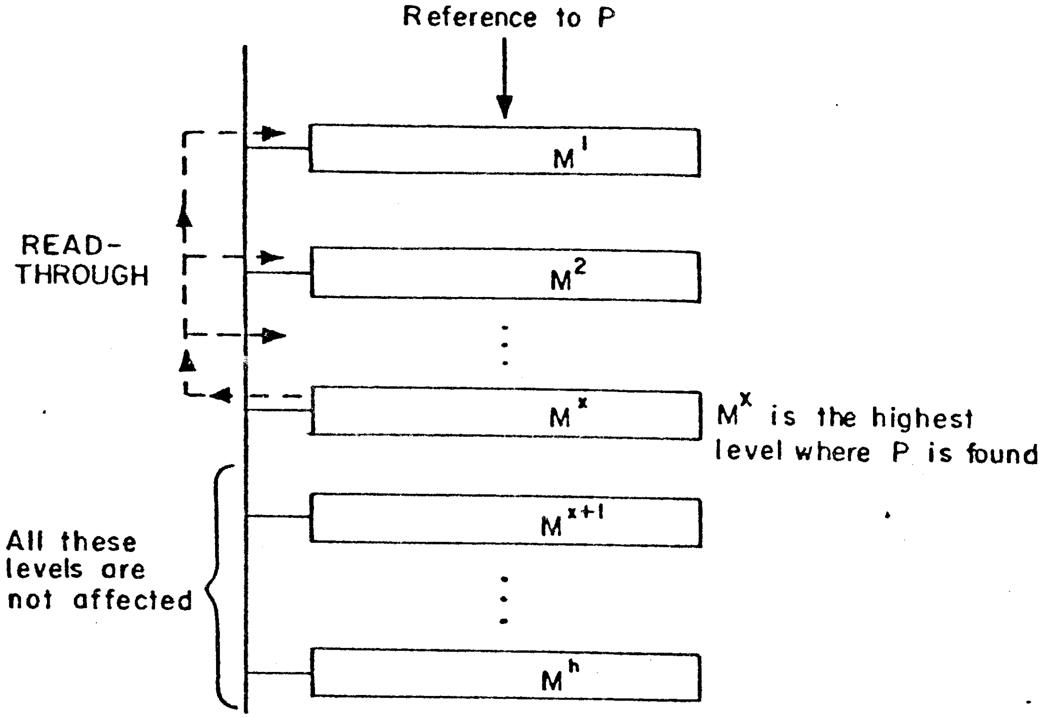


Figure 4.3 Local-LRU Algorithm

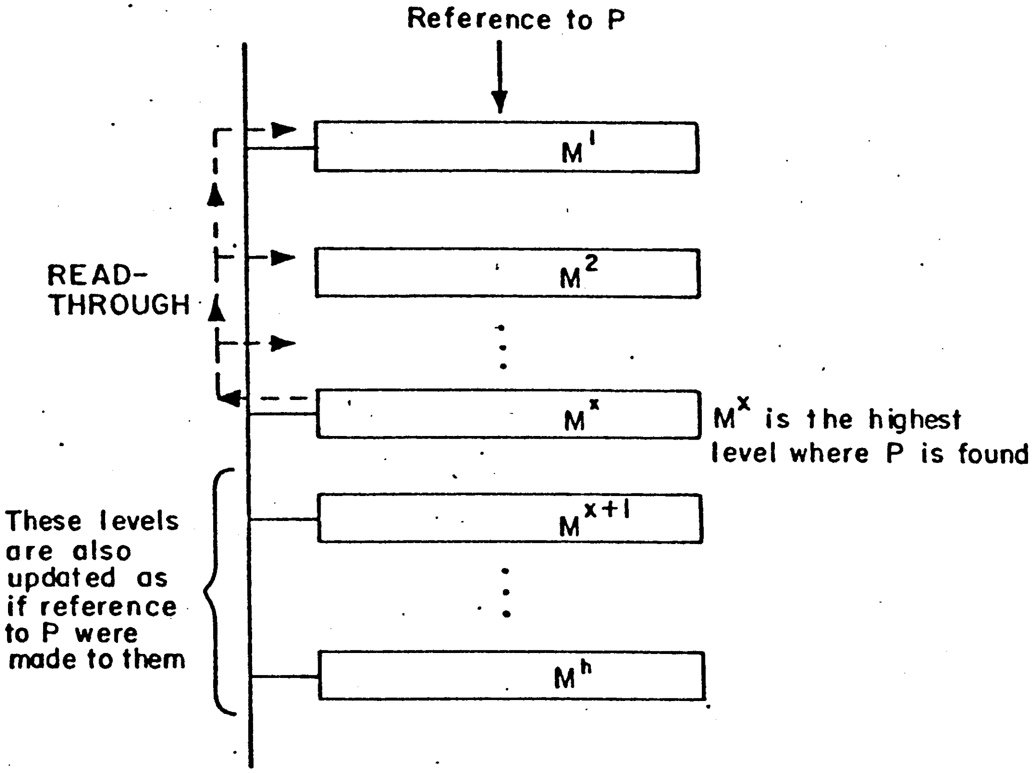

Reference to P
M^1
M^2
READ-THROUGH
M^x
M^x is the highest level where P is found
M^{x+1}
M^h
These levels are also updated as if reference to P were made to them

Figure 4.4 Global-LRU Algorithm

handling overflows generates references which produce overflows, which generate yet more references. Hence another important question to resolve is to determine the conditions under which an overflow from M^i is always found to already exist in M^{i+1}, i.e., no reference to storage levels lower than M^{i+1} is generated as a result of the overflow. This property is called Multi-Level Overflow Inclusion (MLOI). Conditions to guarantee MLOI will be derived in a later section.

We shall consider these important properties in light of four basic algorithm alternatives based on local or global LRU and static or dynamic overflow. Formal definitions for these algorithms will be provided after the basic model of the data storage hierarchy system is introduced.

4.3.2 Basic Model of a Data Storage Hierarchy

For the purposes of this thesis, the basic model illustrated in Figure 4.5 is sufficient to model a data storage hierarchy. As far as the Read-Through and Overflow-Handling operations are concerned, this basic model is generalizable to a h-level storage hierarchy system.

M^r can be viewed as a reservoir which contains all the information. M^i is the top level. It has m_i pages each of size Q_i. M^j (j=i+1) is the next level. It has m_j pages each of size nQ_i where n is an integer greater than 1.

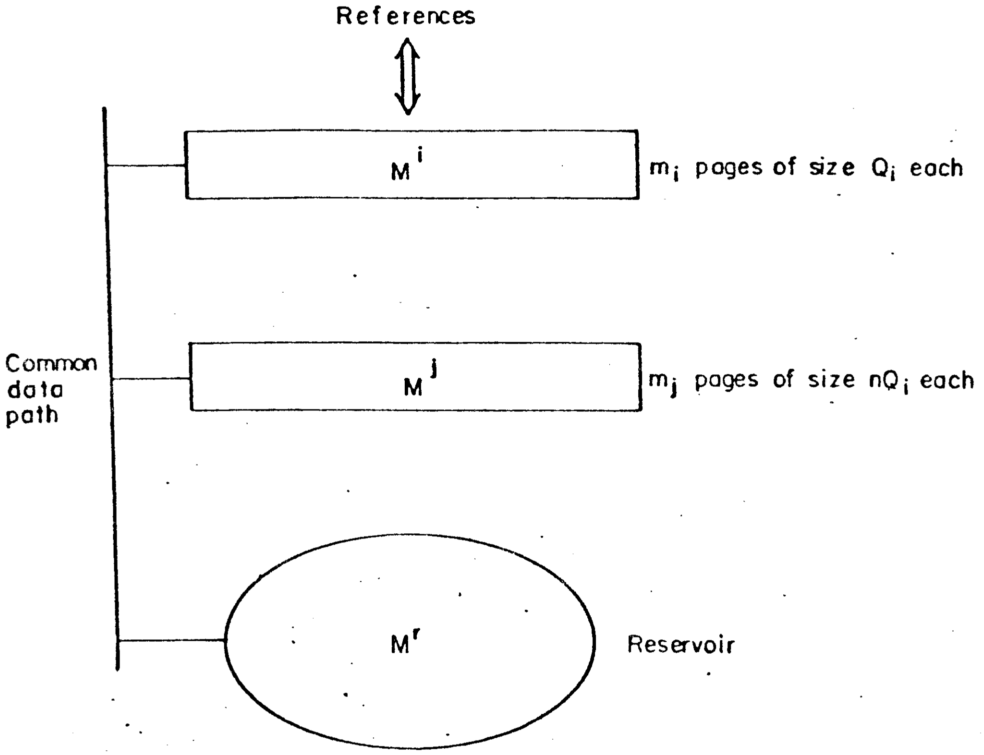

References
M^i
m_i pages of size Q_i each
Common data path
M^j
m_j pages of size nQ_i each
M^r
Reservoir

Figure 4.5 Basic Model of a Data Storage Hierarchy

4.3.3 Formal Definitions of Storage Management Algorithms

Denote a reference string by $r = "r_1, r_2, \ldots, r_n"$, where r_t $(1 \leq t \leq n)$ is the page being referenced at the t-th reference cycle. Let S^i_t be the stack for M^i at the beginning of the t-th reference cycle, ordered according to LRU. That is, $S^i_t = (S^i_t(1), S^i_t(2), \ldots, S^i_t(K))$, where $S^i_t(1)$ is the most recently referenced page and $S^i_t(K)$ is the least recently referenced page. Note that $K \leq m_i$ (m_i = capacity of M^i in terms of the number of pages). The number of pages in S^i_t is denoted as $|S^i_t|$, hence $|S^i_t| = K$. By convention, $S^i_1 = \emptyset$, $|S^i_1| = 0$.

S^i_t is an ordered set. Define M^i_t as the contents of S^i_t without any ordering. Similarly, we can define S^j_t and M^j_t for M^j.

Let us denote the pages in M^j by $P^j_1, P^j_2, \ldots$. Each page, P^j_y, in M^j, consists of an equivalent of n smaller pages, each of size $Q_i = Q_j/n$. Denote this set of pages by $(P^j_y)^i$, i.e., $(P^j_y)^i = \left\{P^i_{y1}, P^i_{y2}, \ldots, P^i_{yn}\right\}$. In general, $(M^j_t)^i$ is the set of pages, each of size Q_i, obtained by "breaking down" the pages in M^j_t. Formally, $(M^j_t)^i = \bigcup_{k=1}^{x} (S^j_t(k))^i$ where $x = |S^j_t|$. $(P^j_y)^i$ is called the family from the parent page P^j_y. Any pair of pages, P^i_{ya} and P^i_{yb} from $(P^j_y)^i$ are said to be family equivalent, denoted by $P^i_{ya} \overset{f}{=} P^i_{yb}$. Furthermore, a parent page P^j_y and a page P^i_{yz} (for $1\underline{<}z\underline{<}n$) from its family are said to be corresponding pages, denoted by $P^i_{yz} \overset{c}{\equiv} P^j_y$.

S^i_t and S^j_t are said to be in corresponding order, denoted by $S^i_t \overset{0}{=} S^j_t$, if $S^i_t(k) \overset{c}{\equiv} S^j_t(k)$ for $k = 1, 2, 3, \ldots w$, where $w = \min (|S^i_t|, |S^j_t|)$. Intuitively, two stacks are in corresponding order if, for each element of the shorter stack, there is a corresponding page in the other stack at the same stack distance (the stack distance for page $S^i_t(k)$ is defined to be k).

M^i_t and M^j_t are said to be correspondingly equivalent, denoted by $M^i_t \overset{e}{\equiv} M^j_t$ if $|M^i_t| = |M^j_t|$ and for any $k = 1, 2, \ldots, |M^i_t|$ there exists x, such that $S^i_t(k) \overset{c}{\equiv} S^j_t(x)$ and $S^j_t(x) \overset{c}{\not\equiv} S^i_t(y)$ for all $y \neq k$. Intuitively, the two memories are correspondingly equivalent when each page in one memory corresponds to exactly one page in the other memory.

The reduced stack, $\bar{S}^i_t$, of S^i_t is defined to be $\bar{S}^i_t(k) = S^i_t(j_k)$ for $k = 1, \ldots, |\bar{S}^i_t|$ where j_k is the minimum j_k where $j_k > j_{k-1} (j_0 = 0)$ and $\bar{S}^i_t(k) \overset{f}{\not\equiv} S^i_t(j)$ for $j < j_k$. Intuitively, $\bar{S}^i_t$ is obtained from S^i_t by collecting one page from each family existing in S^i_t, such that the page being collected from each family is the page that has the smallest stack distance within the family.

In the following, we define the storage management algorithms. In each case, assume that the page referenced at time t is P^i_{ya}.

LRU $(S^i_t, P^i_{ya}) = S^i_{t+1}$ is defined as follows:

Case 1: $P^i_{ya} \in S^i_t,\ P^i_{ya} = S^i_t(k)$:

$$S^i_{t+1}(1) = P^i_{ya},\ S^i_{t+1}(x) = \begin{cases} S^i_t(x-1), & 1 < x \leq k \\ S^i_t(x), & k < x \leq |S^i_t| \end{cases}$$

Case 2: $P^i_{ya} \not\in S^i_t$:

$$S^i_{t+1}(1) = P^i_{ya},\ S^i_{t+1}(x) = S^i_t(x-1),$$

$$1 < x \leq \min(m_i,\ |S^i_t| + 1)$$

If $|S^i_t| = m_i$ then $P^i_{oa} = S^i_t(m_i)$ is the overflow, else there is no overflow.

LOCAL-LRU-SOP $(S^i_t, S^j_t, P^i_{ya}) = (S^i_{t+1}, S^j_{t+1})$ is defined as follows:

Case 1: $P^i_{ya} \in S^i_t$:

$$S^i_{t+1} = \underline{\text{LRU}}\ (S^i_t, P^i_{ya}),\ S^j_{t+1} = S^j_t$$

Case 2: $P^i_{ya} \not\in S^i_t,\ P^j_y \in S^j_t$:

$$S^i_{t'} = \underline{\text{LRU}}\ (S^i_t, P^i_{ya}),\ S^j_{t'} = \underline{\text{LRU}}\ (S^j_t, P^j_y),$$

If there is no overflow from S^i_t

then $S^i_{t+1} = S^i_{t'}$ and $S^j_{t+1} = S^j_{t'}$

If overflow from S^i_t is the page P^i_{oa}

then $(S^i_{t+1}, S^j_{t+1}) = \underline{\text{SOP}}\ (S^i_{t'}, S^j_{t'}, P^i_{oa})$

defined as:

$S^i_{t+1} = S^i_{t'}$; if $P^j_o \in S^j_{t'}$ then $S^j_{t+1} = S^j_{t'}$,

if $P^j_o \not\in S^j_{t'}$ then $S^j_{t+1} = \underline{\text{LRU}}\ (S^j_{t'}, P^j_o)$

Case 3: $P^i_{ya} \not\in S^i_t$ and $P^j_y \not\in S^j_t$:

(handled as in Case 2)

LOCAL-LRU-DOP $(S^i_t, S^j_t, P^i_{ya}) = (S^i_{t+1}, S^j_{t+1})$ is defined as

Case 1: $P^i_{ya} \in S^i_t$:

$S^i_{t+1} = \underline{\text{LRU}}\ (S^i_t, P^i_{ya}),\ S^j_{t+1} = S^j_t$

Case 2: $P^i_{ya} \not\in S^i_t$ and $P^j_y \in S^j_t$:

$S^i_{t'} = \underline{\text{LRU}}\ (S^i_t, P^i_{ya}),\ S^j_{t'} = \underline{\text{LRU}}\ (S^j_t, P^j_y)$

If no overflow from S^i_t then $S^i_{t+1} = S^i_{t'}$

and $S^j_{t+1} = S^j_{t'}$

If overflow from S^i_t is P^i_{oa} then

$(S^i_{t+1}, S^j_{t+1}) = \underline{\text{DOP}}\ (S^i_{t'}, S^j_{t'}, P^i_{oa})$ which is defined as:

$$S^i_{t+1} = S^i_{t'} \text{ and } S^j_{t+1} = \underline{\text{LRU}}\ (S^j_{t'}, P^j_o)$$

Case 3: $P^i_{ya} \not\in S^i_t$ and $P^j_y \not\in S^j_t$:

(handled as in Case 2 above)

GLOBAL-LRU-SOP $(S^i_t, S^j_t, P^i_{ya}) = (S^i_{t+1}, S^j_{t+1})$ is defined as follows:

$S^i_{t'} = \underline{\text{LRU}}\ (S^i_t, P^i_{ya})$ and $S^j_{t'} = \underline{\text{LRU}}\ (S^j_t, P^j_y)$,

If no overflow from S^i_t then $S^i_{t+1} = S^i_{t'}$ and $S^j_{t+1} = S^j_{t'}$

If overflow from S^i_t is P^i_{oa} then (S^i_{t+1}, S^j_{t+1})

$= \underline{\text{SOP}}\ (S^i_{t'}, S^j_{t'}, P^i_{oa})$

$\underline{\text{GLOBAL-LRU-DOP}}\ (S^i_t, S^j_t, P^i_{ya}) = (S^i_{t+1}, S^j_{t+1})$ is defined as

$S^i_{t'} = \underline{\text{LRU}}\ (S^i_t, P^i_{ya})$ and $S^j_{t'} = \underline{\text{LRU}}\ (S^j_t, P^j_y)$

If no overflow from S^i_t then $S^i_{t+1} = S^i_{t'}$ and $S^j_{t+1} = S^j_{t'}$

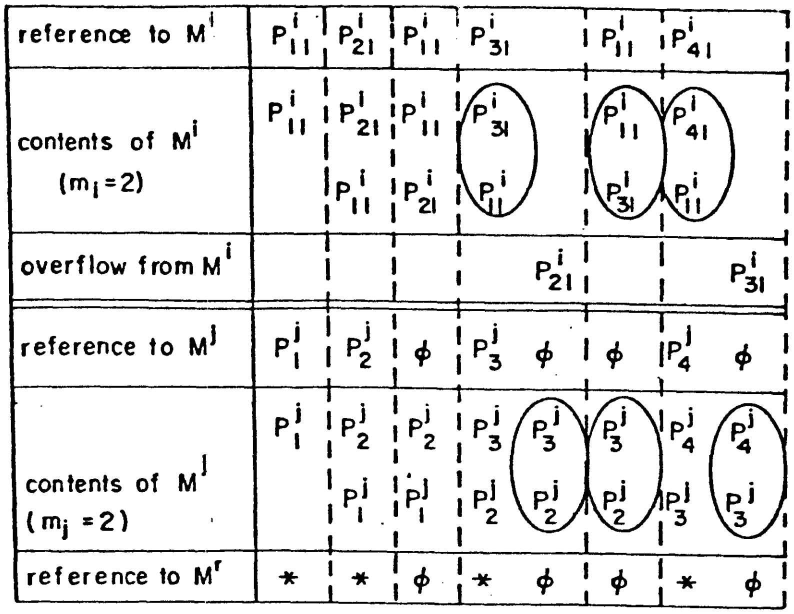
If overflow from S^i_t is P^i_{oa} then $(S^i_{t+1}, S^j_{t+1}) =$

$$\underline{\text{DOP}}\ (S^i_{t'}, S^j_{t'}, P^i_{oa})$$

4.4 PROPERTIES OF DATA STORAGE HIERARCHY SYSTEMS

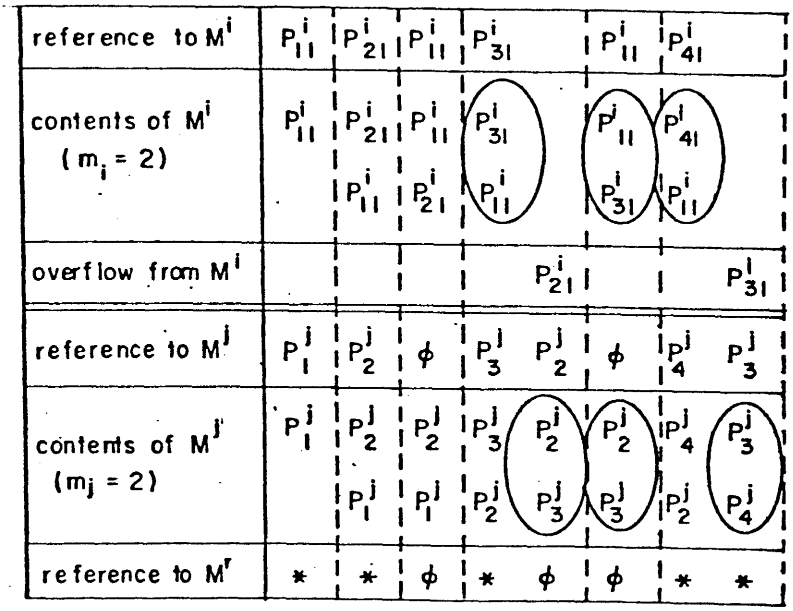
One of the properties of a Read-Through operation is that it leaves a "shadow" of the referenced page (i.e., the corresponding pages) in all storage levels. This provides multiple redundancy for the page. Does this multiple redundancy exist at all times? That is, if a page exists in storage level M^i, will its corresponding pages always be in all storage levels lower than M^i? We refer to this as the Multi-Level Inclusion (MLI) property. As illustrated in Figure 4.6 for the LOCAL-LRU algorithms and in Figure 4.7 for the GLOBAL-LRU algorithms, it is not always possible to guarantee that the MLI property holds. For example, after the reference to P^i_{31} in Figure 4.6(a) the page P^i_{11} exists in M^i but its corresponding page P^j_1 is not found in M^j. In this chapter we shall derive the necessary and sufficient conditions for the MLI property to hold at all times.

Another desirable property of the data storage hierarchy is to avoid generating references due to overflows. That is, under what conditions will overflow pages from M^i find their corresponding pages already existing in the storage level M^{i+1}? We refer to this as the Multi-Level Overflow Inclusion (MLOI) property. We shall investigate the conditions that make this property true at all times.



reference to M^i	P^i_{11}	P^i_{21}	P^i_{11}	P^i_{31}		P^i_{11}	P^i_{41}	
contents of M^i ($m_i=2$)	P^i_{11}	P^i_{21} P^i_{11}	P^i_{11} P^i_{21}	P^i_{31} P^i_{11}		P^i_{11} P^i_{31}	P^i_{41} P^i_{11}	
overflow from M^i					P^i_{21}			P^i_{31}
reference to M^j	P^j_1	P^j_2	ϕ	P^j_3	ϕ	ϕ	P^j_4	ϕ
contents of M^j ($m_j=2$)	P^j_1	P^j_2 P^j_1	P^j_2 P^j_1	P^j_3 P^j_2	P^j_3 P^j_2	P^j_3 P^j_2	P^j_4 P^j_3	P^j_4 P^j_3
reference to M^r	*	*	ϕ	*	ϕ	ϕ	*	ϕ

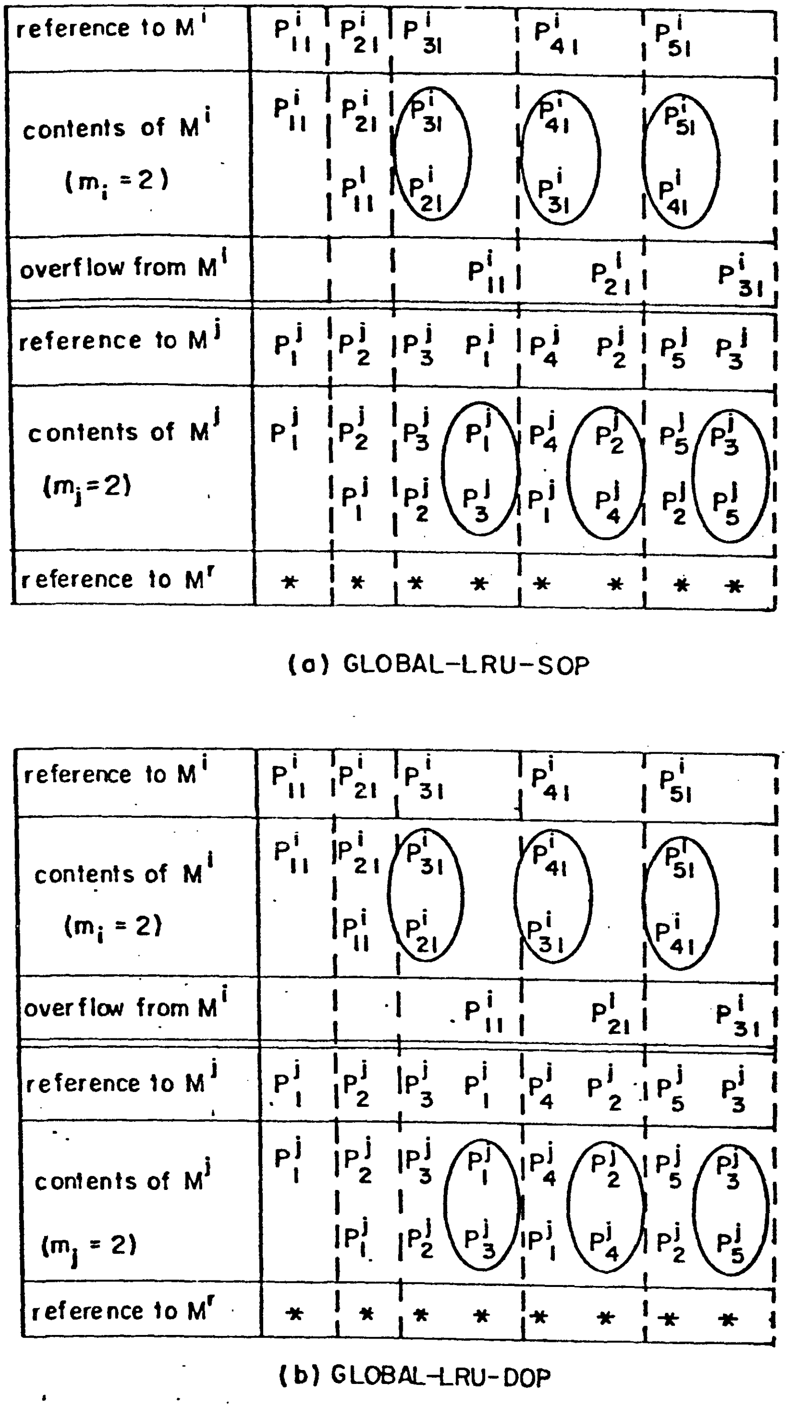
(a) LOCAL-LRU-SOP



reference to M^i	P^i_{11}	P^i_{21}	P^i_{11}	P^i_{31}		P^i_{11}	P^i_{41}	
contents of M^i ($m_i = 2$)	P^i_{11}	P^i_{21} P^i_{11}	P^i_{11} P^i_{21}	P^i_{31} P^i_{11}		P^i_{11} P^i_{31}	P^i_{41} P^i_{11}	
overflow from M^i					P^i_{21}			P^i_{31}
reference to M^j	P^j_1	P^j_2	ϕ	P^j_3	P^j_2	ϕ	P^j_4	P^j_3
contents of M^j ($m_j = 2$)	P^j_1	P^j_2 P^j_1	P^j_2 P^j_1	P^j_3 P^j_2	P^j_2 P^j_3	P^j_2 P^j_3	P^j_4 P^j_2	P^j_3 P^j_4
reference to M^r	*	*	ϕ	*	ϕ	ϕ	*	*

(b) LOCAL-LRU-DOP

Figure 4.6 Violation of MLI by Local-LRU Algorithms



reference to M^i	P^i_{11}	P^i_{21}	P^i_{31}		P^i_{41}		P^i_{51}	
contents of M^i ($m_i = 2$)	P^i_{11}	P^i_{21} P^i_{11}	P^i_{31} P^i_{21}		P^i_{41} P^i_{31}		P^i_{51} P^i_{41}	
overflow from M^i				P^i_{11}		P^i_{21}		P^i_{31}
reference to M^j	P^j_1	P^j_2	P^j_3	P^j_1	P^j_4	P^j_2	P^j_5	P^j_3
contents of M^j ($m_j = 2$)	P^j_1	P^j_2 P^j_1	P^j_3 P^j_2	P^j_1 P^j_3	P^j_4 P^j_1	P^j_2 P^j_4	P^j_5 P^j_2	P^j_3 P^j_5
reference to M^r	*	*	*	*	*	*	*	*

(a) GLOBAL-LRU-SOP

reference to M^i	P^i_{11}	P^i_{21}	P^i_{31}		P^i_{41}		P^i_{51}	
contents of M^i ($m_i = 2$)	P^i_{11}	P^i_{21} P^i_{11}	P^i_{31} P^i_{21}		P^i_{41} P^i_{31}		P^i_{51} P^i_{41}	
overflow from M^i				P^i_{11}		P^i_{21}		P^i_{31}
reference to M^j	P^j_1	P^j_2	P^j_3	P^j_1	P^j_4	P^j_2	P^j_5	P^j_3
contents of M^j ($m_j = 2$)	P^j_1	P^j_2 P^j_1	P^j_3 P^j_2	P^j_1 P^j_3	P^j_4 P^j_1	P^j_2 P^j_4	P^j_5 P^j_2	P^j_3 P^j_5
reference to M^r	*	*	*	*	*	*	*	*

(b) GLOBAL-LRU-DOP

Figure 4.7 Violation of MLI by Global-LRU Algorithms

Referring to the basic model of a data storage hierarchy in Figure 4.5, for high performance it is desirable to minimize the number of references to M^r (the reservoir). If we increased the number of pages in M^i, or in M^j, or in both, we might expect the number of references to M^r to decrease. As illustrated in Figure 4.8 for the LOCAL-LRU-SOP algorithm, this is not always so, i.e., for the same reference string, the number of references to the reservoir actually increased from 4 to 5 after M^i is increased by 1 page in size. We refer to this phenomena as a Multi-Level Paging Anomaly (MLPA). One can easily find situations where MLPA occurs for the other three algorithms. Since occurrence of MLPA reduces performance in spite of the costs of increasing memory sizes, we would like to investigate the conditions to guarantee that MLPA does not exist.

reference to M^i	P^i_{11}	P^i_{21}	P^i_{11}	P^i_{31}		P^i_{11}	P^i_{41}	
contents of M^i ($m_i = 2$)	P^i_{11}	P^i_{21} P^i_{11}	P^i_{11} P^i_{21}	P^i_{31} P^i_{11}		P^i_{11} P^i_{31}	P^i_{41} P^i_{11}	
overflow from M^i					P^i_{21}			P^i_{31}
reference to M^j	P^j_1	P^j_2	ϕ	P^j_3	ϕ	ϕ	P^j_4	ϕ
contents of M^j ($m_j = 2$)	P^j_1	P^j_2 P^j_1	P^j_2 P^j_1	P^j_3 P^j_2	P^j_3 P^j_2	P^j_3 P^j_2	P^j_4 P^j_3	P^j_4 P^j_3
reference to M^r	*	*	ϕ	*	ϕ	ϕ	*	ϕ
number of references to M^r = 4								

reference to M^i	P^i_{11}	P^i_{21}	P^i_{11}	P^i_{31}	P^i_{11}	P^i_{41}	
contents of M^i ($m_i = 3$)	P^i_{11}	P^i_{21} P^i_{11}	P^i_{11} P^i_{21}	P^i_{31} P^i_{11} P^i_{21}	P^i_{11} P^i_{31} P^i_{21}	P^i_{41} P^i_{11} P^i_{31}	
overflow from M^i							P^i_{21}
reference to M^j	P^j_1	P^j_2	ϕ	P^j_3	ϕ	P^j_4	P^j_2
contents of M^j ($m_j = 2$)	P^j_1	P^j_2 P^j_1	P^j_2 P^j_1	P^j_3 P^j_2	P^j_3 P^j_2	P^j_4 P^j_3	P^j_2 P^j_4
reference to M^r	*	*	ϕ	*	ϕ.	*	*
number of references to M^r = 5							

Figure 4.8 Illustration of MLPA

4.4.1 Summary of Properties

The MLI, MLOI, and MLPA properties of the data storage hierarchy have been derived in the form of eight theorems. These theorems are briefly explained and summarized below and formally proven in the following section.

Multi-Level Inclusion (MLI) : It is shown in Theorem 1 that if the number of pages in M^i is greater than the number of pages in M^j (note M^j pages are larger than those of M^i), then it is not possible to guarantee MLI for all reference strings at all times. It turns out that using LOCAL-LRU-SOP, or LOCAL-LRU-DOP, no matter how many pages are in M^j or M^i, one can always find a reference string that violates the MLI property (Theorem 2). Using the GLOBAL-LRU algorithms, however, conditions to guarantee MLI exist. For the GLOBAL-LRU-SOP algorithm, a necessary and sufficient condition to guarantee that MLI holds at all times for any reference string is that the number of pages in M^j be greater than the number of pages in M^i (Theorem 3). For the GLOBAL-LRU-DOP algorithm, a necessary and sufficient condition to guarantee MLI is that the number of pages in M^j be greater than or equal to twice the number of pages in M^i (Theorem 4).

Multi-Level Overflow Inclusion (MLOI) : It is obvious that if MLI cannot be guaranteed then MLOI cannot be guaranteed. Thus, the LOCAL-LRU algorithms cannot guarantee MLOI. For the GLOBAL-LRU-SOP algorithm, a necessary and sufficient condition to guarantee MLOI is the same condition as that to guarantee MLI (Theorem 5). For the GLOBAL-LRU-DOP algorithm, a necessary and sufficient condition to guarantee MLOI is that the number of pages in M^j is strictly greater than twice the number of pages in M^i (Theorem 6). Thus, for the GLOBAL-LRU-DOP algorithm, guaranteeing that MLOI holds will also guarantee that MLI will hold, but not vice versa.

Multi-Level Paging Anomaly (MLPA) : We have identified and proved sufficiency conditions to avoid MLPA for the GLOBAL-LRU algorithms. For the GLOBAL-LRU-SOP algorithm, this condition is that the number of pages in M^j must be greater than the number of pages in M^i before and after any increase in the sizes of the levels (Theorem 7). For the GLOBAL-LRU-DOP algorithm, this condition is that the number of pages in M^j must be greater than twice the number of pages in M^i before and after any increase in the sizes of the levels (Theorem 8).

In summary, we have shown that for the LOCAL-LRU algorithms, no choice of sizes for the storage levels can guarantee that a lower storage level always contains all the information in the higher storage levels. For the

GLOBAL-LRU algorithms, by choosing appropriate sizes for the storage levels, we can (1) ensure that the above inclusion property holds at all times for all reference strings, (2) guarantee that no extra page references to lower storage levels are generated as a result of handling overflows, and (3) guarantee that increasing the sizes of the storage levels does not increase the number of references to lower storage levels. These results are formally stated as the following eight Theorems. Formal proofs of these Theorems are presented in the following section.

THEOREM 1

Under LOCAL-LRU-SOP, or LOCAL-LRU-DOP, or GLOBAL-LRU-SOP, or GLOBAL-LRU-DOP, for any $m_i \geq 2$, $m_j \leq m_i$ implies $\exists$ r,t, $(M_t^j)^i \not\supseteq M_t^i$

THEOREM 2

Under LOCAL-LRU-SOP, or LOCAL-LRU-DOP, for any $m_i \geq 2$, and any m_j, $\exists$ r,t, $(M_t^j)^i \not\supseteq M_t^i$

THEOREM 3

Under GLOBAL-LRU-SOP, for any $m_i \geq 2$, $\forall$ r,t, $(M_t^j)^i \supseteq M_t^i$ iff $m_j > m_i$

THEOREM 4

Under GLOBAL-LRU-DOP, for any $m_i \geq 2$, $\forall$ r,t, $(M_t^j)^i \supseteq M_t^i$ iff $m_j \geq 2m_i$

THEOREM 5

Under GLOBAL-LRU-SOP, for any $m_i \geq 2$, $\forall$ r,t, an overflow from M^i finds its corresponding page in M^j iff $m_j > m_i$

THEOREM 6

Under GLOBAL-LRU-DOP, for any $m_i \geq 2$, $\forall$ r,t, an overflow from M^i finds its corresponding page in M^j iff $m_j > 2m_i$

THEOREM 7

Let M^i (with m_i pages), M^j (with m_j pages) and M^r be System A.

Let M'^i (with m_i' pages), M'^j (with m_j' pages) and M^r be System B.

Let $m_i' \geq m_i$ and $m_j' \geq m_j$. Under GLOBAL-LRU-SOP, for any $m_i \geq 2$, no MLPA can exist if $m_j > m_i$ and $m_j' > m_i'$

THEOREM 8

Let System A and System B be defined as in Theorem 7.

Let $m_i' \geq m_i$ and $m_j' \geq m_j$. Under GLOBAL-LRU-DOP, for any $m_i \geq 2$, no MLPA can exist if $m_j > 2m_i$ and $m_j' > 2m_i'$

4.4.2 Derivation of Properties

THEOREM 1

Under LOCAL-LRU-SOP, or LOCAL-LRU-DOP, or GLOBAL-LRU-SOP, or GLOBAL-LRU-DOP, for any $m_i \geq 2$, $m_j \leq m_i$ implies $\exists\, r,t,\ (M_t^j)^i \not\supseteq M_t^i$

PROOF

Case 1 : $m_j < m_i$

Consider the reference string r=" $P_{1a}^i, P_{2a}^i, \ldots, P_{(m_j+1)a}^i$".

Using any one of the algorithms, the following stacks are obtained at $t=m_j+2$:

$S_t^i = (P_{(m_j+1)a}^i, P_{m_ja}^i, \ldots, P_{2a}^i, P_{1a}^i)$

$S_t^j = (P_{(m_j+1)}^j, P_{m_j}^j, \ldots, P_3^j, P_2^j)$

Thus, $P_{1a}^i \epsilon M_t^i$ but $P_{1a}^i \not\epsilon (M_t^j)^i$, i.e., $(M_t^j)^i \not\supseteq M_t^i$.

Case 2 : $m_j = m_i = w$

Consider the reference string r = " $P_{1a}^i, P_{2a}^i, \ldots, P_{(w+1)a}^i$"

Using any one of the above algorithms, the following stacks are obtained at $t=w+2$:

$S_t^i = (P_{(w+1)a}^i, P_{wa}^i, \ldots, P_{3a}^i, P_{2a}^i)$

$S_t^j = (P_1^j, P_{(w+1)}^j, P_w^j, \ldots P_4^j, P_3^j)$

Thus, $P_{2a}^i \epsilon M_t^i$ but $P_{2a}^i \not\epsilon (M_t^j)^i$, i.e., $(M_t^j)^i \not\supseteq M_t^i$.

Q.E.D.

THEOREM 2

Under LOCAL-LRU-SOP, or LOCAL-LRU-DOP, for any $m_i \geq 2$, and any $m_j, \exists r, t, \ (M_t^j)^i \not\supseteq M_t^i$.

PROOF (For LOCAL-LRU-SOP)

For $m_j \leq m_i$ the result follows directly from THEOREM 1.

For $m_j > m_i$, using the reference string

$r = "\ P_{za}^i, P_{1a}^i, P_{za}^i, P_{2a}^i, \ldots, P_{za}^i, P_{m_j a}^i",$

the following stacks will be produced at $t=2m_j+1$:

$S_t^i = (P_{m_j a}^i, P_{za}^i, P_{(m_j-1)a}^i, \ldots, P_{(m_j-m_i+2)a}^i)$

$S_t^j = (P_{m_j}^j, P_{m_j-1}^j, \ldots, P_2^j, P_1^j)$

Thus, $P_{za}^i \in M_t^i$ but $P_{za}^i \notin (M_t^j)^i$, i.e., $(M_t^j)^i \not\supseteq M_t^i$.

Q.E.D.

PROOF (For LOCAL-LRU-DOP)

For $m_j \leq m_i$ the result follows directly from THEOREM 1.

For $m_j > m_i$, using the following reference string

$r = "\ P_{za}^i, P_{1a}^i, P_{za}^i, P_{2a}^i, \ldots, P_{za}^i, P_{m_j a}^i",$

The following stacks will be produced at $t=2m_j+1$:

$S_t^i = (P_{m_j a}^i, P_{za}^i, \ldots, P_{(m_j-m_i+2)a}^i),$

$S_t^j = (a_1, a_2, \ldots, a_{m_j})$

Where for $1 \leq i \leq m_j$, $a_i \in \left\{P_{m_j}^j, P_{m_{j-1}}^j, \ldots, P_3^j, P_2^j, P_1^j\right\}$

since P_z^j is the only overflow from M^j.

Thus, $P_{za}^i \in M_t^i$ but $P_{za}^i \notin (M_t^j)^i$, i.e., $(M_t^j)^i \not\supseteq M_t^i$.

Q.E.D.

THEOREM 3

Under GLOBAL-LRU-SOP, for any $m_i \geq 2$, $\forall$ r,t, $(M^j_t)^i \supseteq M^i_t$ iff $m_j > m_i$

PROOF

This proof has two parts. Part (a) to prove $\forall$ r,t, $(M^j_t)^i \supseteq M^i_t \Rightarrow m_j > m_i$

or equivalently, $m_j \leq m_i \Rightarrow \exists r,t, (M^j_t)^i \not\supseteq M^i_t$

Part (b) to prove $m_j > m_i \Rightarrow \forall r,t, (M^j_t)^i \supseteq M^i_t$

PROOF of Part (a): $m_j \leq m_i \Rightarrow \exists r,t, (M^j_t)^i \not\supseteq M^i_t$

This follows directly from THEOREM 1.

Q.E.D.

To Prove Part (b), we need the following results.

LEMMA 3.1

$\forall$ r,t such that $|M^j_t| \leq m_i$, if $m_j = m_i + 1$, then (a) $(M^j_t)^i \supseteq M^i_t$, and (b) $\bar{S}^i_t \overset{0}{=} S^j_t$

PROOF of LEMMA 3.1

For t=2 (i.e., after the first reference), (a) and (b) are true. Suppose (a) and (b) are true for t, such that $|M^j_t| \leq m_i$

Consider the next reference:

Case 1: It is a reference to M^i:

There is no overflow from M^i or M^j, so (a) is still true. Since Global-LRU is used, (b) is still true.

Case 2: It is a reference to M^j:

There is no overflow from M^j. If no overflow from M^i, the same arguement as Case 1 applies. If there is overflow from M^i, the overflow page finds its corresponding page in M^j. Since SOP is used, this overflow can be treated as a "no-op". Thus (a) and (b) are preserved.

Case 3: It is a reference to M^r:

There is no overflow from M^j since $|M^j_{t+1}| \leq m_i$. Thus the same reasoning as in Case 2 applies.

Q.E.D.

LEMMA 3.2

$\forall$ r,t, such that $|M^j_t| = m_j$, if $m_j = m_i+1$ then

(a) $(M^j_t)^i \supset M^i_t$, (b) $\overline{S}^i_t \overset{\cap}{=} S^j_t$, and (c) $(S^j_t(m_j))^i \cap S^i_t = \emptyset$

Let us denote the conditions (a) (b) and (c) jointly as Z(t).

PROOF of LEMMA 3.2

Suppose the first time $S^j_t(m_j)$ is filled is by the t*-th reference. That is, $S^j_t(m_j) = \emptyset$ for all $t \leq t^*$ and $S^j_t(m_j) \neq \emptyset$ for all $t > t^*$. From LEMMA 3.1 we know that (a) and (b) are true for all $t \leq t^*$.

Let $t_1 = t^* + 1$, $t_2 = t^* + 2$, ..., etc. We shall show, by induction on t, starting at t_1, that A(t) is true. First we show that $Z(t_1)$ is true as follows:

$\underline{\text{Case 1}}$: $M^j_{t*} \underline{\underline{e}} M^i_{t*}$

$\overline{S}^i_{t*} \underline{\underline{0}} S^j_{t*}$ and $M^j_{t*} \underline{\underline{e}} M^i_{t*} \Rightarrow S^j_{t*}(m_j-1) \underline{\underline{c}} S^i_{t*}(m_i)$
As a result of the reference at t* (to M^r),

$S^j_{t*+1}(m_j) = S^j_{t*}(m_j-1)$ and $S^i_{t*}(m_i)$ overflows from M^i. This overflow page finds its corresponding apge in M_j because there is no overflow from M^j and (a). Since SOP is used, the overflow from M^i can be treated as a "no-op". Furthermore, since Global-LRU is used, (b) is true after the t*-th reference, (b) and $|S^j_{t*+1}| > |S^i_{t*+1}| \Rightarrow$(a) and (c). Thus $Z(t_1)$ is true.

$\underline{\text{Case 2}}$: $(M^j_{t*})^i \supset M^i_{t*}$ and M^j_{t*} and $\not\underline{\underline{e}} M^i_{t*}$

$(M^j_{t*})^i \supset M^i_{t*}$ and $M^j_{t*} \not\underline{\underline{e}} M^i_{t*} \Rightarrow \exists S^j_{t*}(k)$ such that $(S^j_{t*}(k))^i \cap M^i_{t*} = \emptyset$ $\overline{S}^i_{t*} \underline{\underline{o}} S^j_{t*}$ and $(S^j_{t*}(k))^i \cap M^i_{t*} = \emptyset$ $k > |\overline{S}^i_{t*}|$and $(S^j_{t*}(x))^i \quad M^i_{t*} = \emptyset$ for all x where $m_{j-1} \geq x \geq k$. Thus $(S^j_{t*}(m_{j-1}))^i \cap S^i_{t*} = \emptyset$ (i.e., the last page of S^j_{t*} is not S^i_{t*})

$S^i_{t*}(m_i)$ overflows from M^i. There is no overflow from M^j. Thus the overflow page from M^i finds its corresponding page in M^j. For the same reasons as in Case 1, (b) is still preserved. (b) and $|S^j_{t*+1}| > |S^i_{t*+1}| \Rightarrow$

(a) and (c) are true. Thus, $Z(t_1)$ is true.
Assume that $Z(t_k)$ is true; to show that $Z(t_{k+1})$ is true, we consider the next reference, at time t_{k+1}:

Imagine that the last page of $S^j_{t_k}$ does not exist, i.e., $S^j_{t_k}(m_j) = \emptyset$. If the reference at t_{k+1} is to a page in $M^i_{t_k}$ or $M^j_{t_k}$, then (a) and (b) still hold because Global-LRU is used and because overflow from M^i finds its corresponding page in M^j (See the proof of LEMMA 3.1).

If the reference at t_{k+1} is to a page not in $M^j_{t_k}$, then we can apply the agruement as that used in considering the reference at time t_1 above to show that $Z(t_{k+1})$ is still true.

Q.E.D.

LEMMA 3.3

$\forall$ r,t, if $m_j=m_i+1$ then (a) $(M^j_t)^i \supseteq M^i_t$ and (b) $(S^j_t(m_m))^i \cap S^i_t = \emptyset$

PROOF of LEMMA 3.3

For t such that $|M^j_t| \leq m_i$ (a) follows directly from LEMMA 3.1 and (b) is true because $S^j_t(m_j) = \emptyset$

For t such that $|M^j_t| = m_j$ (a) and (b) follows directly from LEMMA 3.2

Q.E.D.

LEMMA 3.4

$\forall$ r,t, if $m_j > m_i$ than (a) $(M^j_t)^i \supseteq M^i_t$ and (b) $(S^j_t(m_j))^i \cap S^i_t = \emptyset$

PROOF of LEMMA 3.4

Let $m_j = m_i+k$. We shall prove this lemma by induction of k. For k=1 (a) and (b) are true from LEMMA 3.3. Suppose that (a) and (b) are true for k. Consider $m_j=m_i+(k+1)$. That is consider the effects of increasing M^j by 1 page in size:

Since M^i is unchanged, M^j (with m_i+k+1 pages) sees the same reference string as M^j (with m_i+k pages). Applying the stack inclusion property (Mattson et al., 1970), we have M^j(with m_i+k+1 pages) $\supseteq M^j$ (with m_i+k pages). Thus (a) is still true. Suppose $(S_t^j(m_i+k+1))^i \cap S_t^i \neq \emptyset$ then there is a page in M^i that corresponds to this page. But $S_t^j(m_i+k+1)$ is not in M^j (with m_i+k pages). This contradicts the property that $(M_t^j)^i \supseteq M_t^i$. This showes that (b) is still true.

Q.E.D.

PROOF of Part (b): $m_j > m_i \Rightarrow \forall$ r,t, $(M_t^j)^i \supseteq M_t^i$:

This follows directly from LEMMA 3.4.

Q.E.D.

THEOREM 4

Under GLOBAL-LRU-DOP, for any $m_i \geq 2$, $\forall$ r,t,$(M_t^j)^i \supseteq M_t^i$ iff $m_j \geq 2m_i$

PROOF

This proof has two parts:

Part (a): $m_j < 2m_i \Rightarrow \exists$ r,t, $(M_t^j)^i \not\supseteq M_t^i$

Part (b): $m_j \geq 2m_i \Rightarrow \forall$ r,t, $(M_t^j)^i \supseteq M_t^i$

PROOF of Part (a): $m_j < 2m_i \Rightarrow \exists$ r,t,$(M_t^j)^i \not\supseteq M_t^i$

For $m_j \leq m_i$ the result follows from THEOREM 1.

Consider the case for $2m_i > m_j > m_i$:

The reference string r = "P^i_{1a} , P^i_{2a}, P^i_{3a}, ...,$P^i_{(2m_i)a}$"

will produce the following stacks:

$S_t^i = (P^i_{(2m_i)a}, P^i_{(2m_i-1)a}, \ldots, P^i_{(m_i+1)a})$,

$S_t^j = (a_1, a_2, a_3, \ldots, a_{m_j})$ where a_i's are picked from L_1 and L_2 alternatively, starting from L_1.

$L_1 = (P^j_{m_i}, P^j_{(m_i-1)}, \ldots, P^j_1)$ and

$L_2 = (P^j_{2m_i}, P^j_{(2m_i-1)}, \ldots, P^j_{m_i+1})$.

If m_j is even, then $(a_1, a_3, \ldots a_{m_j-1})$ corresponds to the frist $m_j/2$ elements of L_1 and $(a_2, a_4, \ldots a_{m_j})$ corresponds to the first $m_j/2$ elements in L_2. We see that $P^i_{(m_i+1)a}$ is in S_t^i but its corresponding page is not in S_t^j ($P^j_{(m_i+1)}$ is not in S_t^j since $m_j/2 < m_i$).

If m_j is odd, then $(a_1, a_3, \ldots a_{m_j})$ corresponds to the first $(m_j+1)/2$ elements in L_1 and $(a_2, 2_4, \ldots, a_{m_j-1})$

corresponds to the first $(m_j-1)/2$ elements in L_2. We see that the page $P^i_{(m_i+1)a}$ is in S^i_t but its corresponding page is not in S^j_t because $\max(\ (m_j-1)/2) = m_i-1$, thus, $a_{(m_j-1)}$ is at most the (m_i-1)-th element of L_2, $P^j_{2m_i-(m_i-1)+1}=P^j_{m_i+2}$. In both cases, $(M^j_t)^i \not\supseteq M^i_t$.

Q.E.D.

To prove Part (b), we need the following preliminary results.

LEMMA 4.1

Under GLOBAL-LRU-DOP, for $m_i \geq 2$, $m_j \geq 2m_i$, a page found at stack distance k in M^i_t implies its corresponding page can be found within stack distance 2k in M^j_t.

PROOF of LEMMA 4.1

We prove by induction on t.

At t=1, the statement is trivially true. At t=2 (i.e., after the first reference) $S^i_t(1)$ and its corresponding page are both at the beginning of the stack, hence the induction statement is still true.

Suppose the induction statement is true at time t, i.e., $P^i_{za} = S^i_t(k) \Rightarrow P^j_z$ can be found within stack distance 2k within S^j_t.

Suppose the next reference is to P^i_{wa}. There are three cases:

Case 1: $P^i_{wa} \varepsilon M^i_t$ $(P^i_{wa}=S^i_t(x))$

From the induction statement, P^j_w is found within stack distance 2k in S^j_t as illustrated in Fugure 4.9. Consider the page movements in the two stacks as a result of handling the reference to P^i_{wa}:

(1) P^i_{wa} and P^j_w are both moved to the top of their stack, the induction statement still holds for these pages.

(2) Each page in A increases its stack distance by 1, but its corresponding page is in A', each page of which can at most increase its stack distance by 1. Thus the induction statement holds for all pages in A.

(3) None of the pages in B are moved. None of the pages in B' are moved. (See Figure 4.9) If a page in B has its corresponding page in B', the induction statement is not violated. Suppose a page in B, $P^i_{ba} = S^i_t(k)$ $(k>x)$, has its corresponding page, $P^j_b = S^j_t(w)$ in A'. Then P^j_b can at most increase its stack distance by 1. But $w \leq 2x$ because $P^j_b \varepsilon A'$. Since $2k > 2x$, the induction statement is not violated.

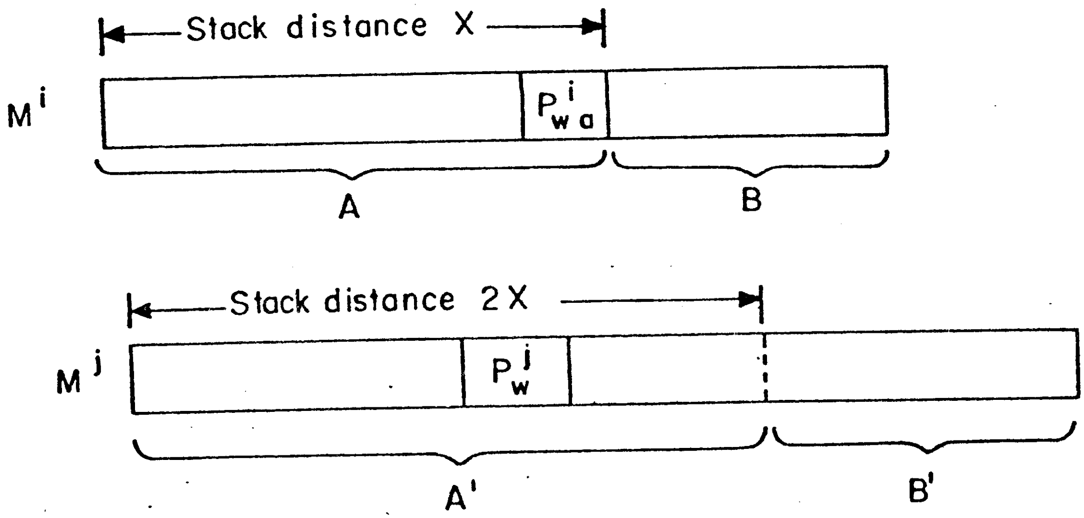


Figure 4.9

Case 2: $P^i_{wa} \not\in M^i_t$, $P^j_w \in M^j_t$

Each page in M^i increases its stack distance by 1. Each corresponding page in M^j can at most increase its stack distance by 2, one due to the reference and one due to an overflow from M^i. Hence if $P^j_{za} = S^i_t(k)$, $k < m_i$, then $P^i_{za} = S^i_{t+1}(k+1)$, and P^j_z can be found within stack distance $2(k+1)$ in M^j at time $t+1$.

Case 3: $P^i_{wa} \not\in M^i_t$, $P^j_w \not\in M^j_t$

As a result of the read-through from M^r, each page in M^i is increased by a stack distance of 1. That is, for $k < m_i$, $P^i_{za} = S^i_t(k) \Rightarrow P^i_{za} = S^i_{t+1}(k+1)$.
Each page in M^j can at most increase its stack distance by 2, one due to loading the referenced page and one due to an overflow from M^i. Hence, the page P^j_z is found within stack distance of $2k+2$ in M^j. Since $\max(2k+2) = 2m_i \leq m_j$, P^j_z is still in M^j.

Q.E.D.

COROLLARY to LEMMA 4.1

$m_j > 2m_i \Rightarrow \forall\ r,t,\ (S^j_t(m_j))^i \cap S^i_t = \emptyset$

PROOF of COROLLARY

For any P^i_{za} in S^i_t, its corresponding page can be found within stack distance $2m_i$ in S^j_t, and since pages in S^j_t are unique, the information in the last page of S_t^j is not found in S_t^i, i.e., $(S_t^j(m_j))^i \cap S_t^i = \emptyset$.

PROOF of Part (b): $m_j \geq 2m_i \Rightarrow \forall\ r,t,\ (M_t^j)^i \supseteq M_t^i$

This follows directly from LEMMA 4.1.

Q.E.D.

THEOREM 5

Under GLOBAL-LRU-SOP, for any $m_i \geq 2$, $\forall$ r,t, an overflow from M^i finds its corresponding page in M^j iff $m_j > m_i$

COROLLARY

Under GLOBAL-LRU-SOP, for any $m_i \geq 2$, $\forall$ r,t, an overflow from M^i finds its corresponding page in M^j iff $\forall$ r,t, $(M_t^j)^i \supseteq M_t^i$.

PROOF

This Proof has two parts as shown below.

PROOF of Part (a): $m_j > m_i \Rightarrow \forall$ r,t, an overflow from M^i finds its corresponding page in M^j

From LEMMA 3.4 $m_j > m_i \Rightarrow \forall$ r,t, $(M_t^j)^i \supseteq M_t^i$ and $(S_t^j(m_j))^i \cap S_t^i = \emptyset$. Suppose the overflow from M^i, P_{oa}^i is caused by a reference to M^j. Then just before P_{oa}^i is overflowed, P_o^j exists in M^j. After the overflow, P_{oa}^i finds its corresponding page still existing in M^j. Suppose the overflow, P_{oa}^i , is caused by a reference to M^r. Then just before the overflow from M^i, P_o^j exists in M^j and $(S_t^j(m_j))^i \cap S_t^i = \emptyset$ i.e., the information in the last page of M^j is not M^i. This means that the last page of M^j is not P_o^j, thus, the overflow page P_{oa}^i finds its corresponding page still in M^j after an overflow from M^j occurs.

PROOF of Part (b): $m_j \leq m_i \Rightarrow \exists$ r,t, such that an overflow from M^i does not find its corresponding page in M^j.

From THEOREM 1, $m_j \leq m_i \Rightarrow \exists$ r,t, $(M_t^j)^i \not\supseteq M_t^i$, then there exists $P_{za}^i \varepsilon M_t^i$ and $P_z^j \not\varepsilon M_t^j$. We can find a reference string such that at the time of the overflow of P_{za}^i from M^i, P_z^j is still not in M^j. A string of references to M^r will produce this condition. Then at the time of overflow of P_{za}^i, it will not find its corresponding page in M^j.

Q.E.D.

THEOREM 6

Under GLOBAL-LRU-DOP, for $m_i \geq 2$, $\forall$ r,t, an overflow from M^i finds its corresponding page in M^j iff $m_j > 2m_i$

COROLLARY

Under GLOBAL-LRU-DOP, for $m_i \geq 2$, $\forall$ r,t, an overflow from M^i finds its corresponding page in M^j implies that $\forall$ r,t, $(M_t^j)^i \supseteq M_t^i$.

PROOF

This proof has two parts as shown below.

PROOF of Part (a): $m_j > 2m_i \Rightarrow \forall$ r,t, an overflow from M^i finds its corresponding page in M^j.

THEOREM 4 ensures that $m_j > 2m_i \Rightarrow \forall$ r,t, $(M_t^j)^i \supseteq M_t^i$ and LEMMA 4.1 ensures that $(S_t^j(m_j))^i \cap S_t^i = \emptyset$, we then use the same argument as in Part (a) of THEOREM 5.

PROOF of Part (b): $m_j \leq 2m_i \Rightarrow \exists$ r,t, such that an overflow from M^i does not find its corresponding page in M^j.

Case 1: $m_j < 2m_i$

$m_j < 2m_i \Rightarrow \exists$ r,t, $(M_t^j)^i \not\supseteq M_t^i$ (from the proof of part (a) of THEOREM 4). We then use the same argument as in Part (b) of THEOREM 5.

Case 2: $m_j = 2m_i$

The reference string r = "P_{1a}^i, P_{2a}^i, ..., $P_{(2m_i)a}^i$, $P_{(2m_i+1)a}^i$" will produce the following stacks (at $t=2m_i+1$):

$S_t^i = (P^i_{(2m_i)a}, P^i_{(2m_i-1)a}, \ldots, P^i_{(m_i+1)a})$

$S_t^j = (P^j_{m_i}, P^j_{2m_i}, P^j_{m_i-1}, P^j_{2m_i-1}, \ldots, P^j_1, P^j_{m_i+1})$

In handling the next reference, to page $P^i_{(2m_i+1)a}$, the pages $P^i_{(m_i+1)a}$ and $P^j_{m_i+1}$ overflow at the same time, hence the overflow page $P^i_{(m_i+1)a}$ from M^i does not find its corresponding page in M^j.

Q.E.D.

THEOREM 7

Let M^i (with m_i pages), M^j (with m_j pages) and M^r be System A. Let M'^i (with m_i' pages), M'^j (with m_j' pages) and M^r be System B. Let $m_i' \geq m_i$ and $m_j' \geq m_j$. Under GLOBAL-LRU-SOP, for any $m_i \geq 2$, no MLPA can exist if $m_j > m_i$ and $m_j' \geq m_i'$.

PROOF

We shall show that $\forall$ r,t, $(M_t^i \cup (M_t^j)^i) \subseteq (M'^i_t \cup (M'^j_t)^i)$ This will ensure that no MLPA can exist.

Since $m_i' \geq m_i$ and LRU is used in M^i and M'^i, we can apply the LRU stack inclusion property to obtain $M_t^i \subseteq M'^i_t$.

From THEOREM 5, we know that overflows from M^i or from M'^i always find their corresponding pages in M^j and M'^j respectively. Since SOP is used, these overflows can be treated as "no-ops". Thus, M^j and M'^j see the same reference string and we can apply the LRU stack inclusion property to obtain $M_t^j \subseteq M'^j_t$ (since $m_j' \geq m_j$ and LRU is used).

$M_t^i \subseteq M'^i_t$ and $M_t^j \subseteq M'^j_t \Rightarrow (M_t^i \cup (M_t^j)^i) \subseteq (M'^i_t \cup (M'^j_t)^i)$.

Q.E.D.

THEOREM 8

Let System A and System B be defined as in THEOREM 7. Let $m_i' \geq m_i$ and $m_j' \geq m_j$. Under GLOBAL-LRU-DOP, for any $m_i \geq 2$, no MLPA can exist if $m_j > 2m_i$ and $m_j'>2m_i'$.

PROOF

We need the following preliminary results for this proof.

LEMMA 8.1

Let S_t^j be partitioned into two disjoint stacks, W_t and V_t defined as follows: $W_t(k) = S_t^j(j_k)$ for $k=1,\ldots,|W_t|$ where $j_0=0$, and j_k is the minimum $j_k>j_{k-1}$ such that $\exists P_{za}^i \epsilon S_t^i$ and $P_{za}^i \overset{c}{\equiv} S_t^j(j_k)$.

$V_t(k) = S_t^j(j_k)$ for $k=1, \ldots, |V_t|$ where $j_0=0$, and j_k is the minimum $j_k > j_{k-1}$ such that $\forall P_{za}^i \epsilon S_t^i$, $P_{za}^i \overset{c}{\not\equiv} S_t^j(j_k)$. (Intuitively, W_t is the stack obtained from S_t^j by collecting those pages that have their corresponding pages in M_t^i such that the order of these pages in S_t^j is preserved. V_t is what is left of S_t^j after W_t is formed.) Then, $\forall r,t$, (a) $W_t \overset{e}{\equiv} \bar{S}_t^i$ and (b) $V_t \subseteq O_t$ where O_t is the set of pages corresponding to all the pages that ever overflowed from M^i, up to time t.

PROOF of LEMMA 8.1

From THEOREM 4, $m_j > 2m_i \Rightarrow \forall r,t,\ (M_t^j)^i \supseteq M_t^i$. Thus, for each page in M_t^i, its corresponding page is in M_t^j.

This set of pages in M_t^j is exactly W_t, and $W_t \subseteq \bar{S}_t^i$ by definition. Since the conditions for V_t and W_t are mutually exclusive and collectively exhaustive, the other pages in M_t^j that are not in W_t are by definition in V_t. Since a page in V_t does not have a corresponding page in M_t^i, its corresponding page must have once been in M^i because of Read-Through, and later overflowed from M^i. Thus a page in V_t is a page in O_t.

Q.E.D.

LEMMA 8.2

Any overflow page from M_t^j is a page in V_t

PROOF of LEMMA 8.2

From THEOREM 4, $m_j > 2m_i \Rightarrow \forall r,t, (M_t^j)^i \supseteq M_t^i$

From THEOREM 6, $m_j > 2m_i \Rightarrow \forall r,t,$ an overflow from M^i always finds its corresponding page in M^j

An overflow from M_t^j is caused by a reference to M^r. An overflow from M_t^j also implies that there is an overflow from M_t^i.

Suppose the overflow page from M_t^j is P_o^j. Also suppose $P_o^j \varepsilon W_t$, i.e., $P_o^j \not\subseteq V_t$. We shall show that this leads to a contradiction.

The overflow page from M_t^i is either P_{oa}^i or $P_{ya}^i (y \neq o)$. If $P_{oa}^i \subseteqq P_o^j$ is overflowed from M_t^i, THEOREM 6 is violated since P_{oa}^i and P_o^j overflow at the same time so P_{oa}^i will not find its corresponding page in M^j.

If $P_{ya}^i \subsetneqq P_o^j$ is overflowed from M_t^i, THEOREM 4 is violated since after the overflow handling, there exists a page $P_{ob}^i \subseteqq P_o^j$ in M^i (since $P_o^j \varepsilon W_t$) but P_o^j is no longer in M^j.

Q.E.D.

LEMMA 8.3

If there is no overflow from either M^j or M'^j then $\forall$ r,t, V_t and V_t' have the same reverse ordering.

Two stacks S^i and S^j are in the same reverse ordering,

$S^i \; \underline{\underline{ro}} \; S^j$, if $rS^i(k) = rS^j(k)$ for $1 \le k \le \min\ (|S^i|\ ,\ |S^j|)$, where rS denotes the stack obtained from S by reversing its ordering. By convention, $S^i \; \underline{\underline{ro}} \; S^j$ if $S^i = \emptyset$ or $S^j = \emptyset$

PROOF of LEMMA 8.3

To facilitate the proof, we introduce the following definitions:

(1) The ordered parent stack, $(S^i)^j$, of the stack S^i is the stack of parent pages corresponding to, and in the same ordering as, the pages in the reduced stack, $\bar{S}^i$, of S^i. Formally, $(S^i)^j \; \underline{\underline{e}} \; \bar{S}^i$ and $(S^i)^j \; \underline{\underline{o}} \; \bar{S}^i$

(2) Define a new binary operator, concatenation ($||$), between two stacks, S^1 and S^2, to produce a new stack, S, as follows:

$$S = S^1||S^2, \text{ where } S(k) = \begin{cases} S^1(k) & \text{for } k=1,\ 2,\ldots,|S^1| \\ S^2(k) & \text{for } k=|S^1|+1,\ldots, \\ & (|S^1| + |S^2|) \end{cases}$$

(3) Define a new binary operator, ordered difference ($\underline{o}$), between a stack S^1 and a set T, to produce a new stack, S, as follows:

$$S = S^1 \; \underline{o} \; T, \text{ where } S(k) = S^1(j_k) \text{ for } k=1,2,\ldots, (|S^1| - |S^1 \cap T|\ ),$$

such that $j_0 = 0$, j_k is the minimum $j_k > j_{k-1}$ such that $S^1(j_k) \cap T = \emptyset$. Intuitively, S is obtained from S^1 by taking away those elements of S^1 which are also in T.

Figure 4.10 illustrates the LRU ordering of all Level i pages ever referenced up to time t. Since there is no overflow from either M^j of M'^j, the length of this LRU stack is less than or equal to $\min(m_j, m_j')$

By the definition of V_t', $V_t' = (Y_t)^j \underline{o} (S'^i_t)^j$

But $(S'^i_t)^j = (S^i_t)^j \,||\, ((X_t)^j \underline{o} (S^i_t)^j)$,

hence $V_t' = (Y_t)^j \underline{o} ((S^i_t)^j \,||((X_t)^j \underline{o} (S^i_t)^j))$

$$= (Y_t)^j \underline{o} ((S^i_t)^j \cup (X_t)^j)$$

Similarly, by the definition of V_t, $V_t = (Z_t)^j \underline{o} (S^i_t)^j$

But $(Z_t)^j = (X_t)^j \,||\, ((Y_t)^j \underline{o} (X_t)^j)$,

Hence $V_t = ((X_t)^j \underline{o} (S^i_t)^j) \,||\, (((Y_t)^j \underline{o} (X_t)^j) \underline{o} (S^i_t)^j)$

$$= ((X_t)^j \underline{o} (S^i_t)^j) \,||\, ((Y_t)^j \underline{o} ((S^i_t)^j \cup (X_t)^j))$$

$$= ((X_t)^j \underline{o} (S^i_t)^j) \,||\, V_t'$$

Thus, the two stacks are in the same reverse ordering.

Q.E.D.

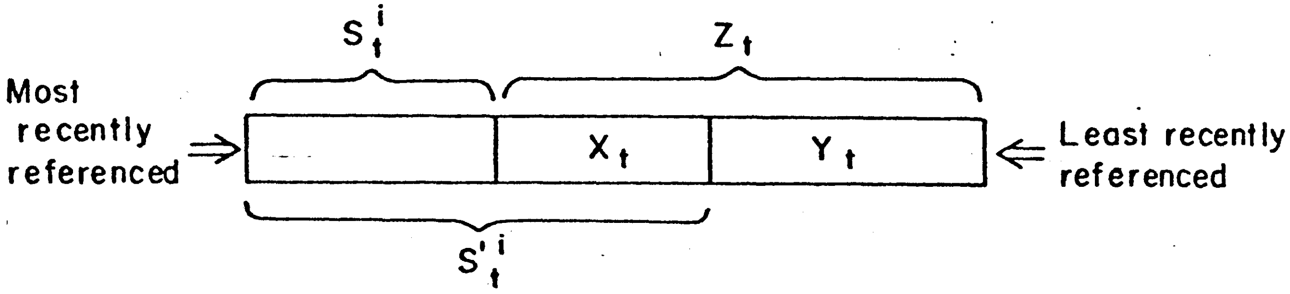

S_t^i
Z_t
Most recently referenced
X_t
Y_t
Least recently referenced
S'^i_t

Figure 4.10

LEMMA 8.4

$\forall$r,t, (a) $M'^j_t \supseteq M^j_t$, (b) V_t and V'_t are either in the same reverse ordering or the last element of V'_t is not an element of V_t

PROOF of LEMMA 8.4

(a) and (b) are true for any time before there is any overflow from either M^j or M'^j. (a) is true because any page ever referenced is in Level j, so a page found in M^j is also found in M'^j. (b) is true because of the result from LEMMA 8.3.

Assume that (a) and (b) is true for t. Consider the next reference at t+1. Suppose this reference does not produce any overflow from either M^j or M'^j, then (a) still holds because $M'^j_t \supseteq M^j_t$ and $M'^i_t \supseteq M^i_t$ (See THEOREM 7). (b) still holds because overflows from M^j and M'^j are taken from the end of stacks V_t and V'_t respectively, and since there is no overflow from Level j, (b)'s validity is not disturbed.

Suppose this reference does produce overflow(s) from Level j.

Case 1 : overflow from M'^j, no overflow from M^j :

This cannot happen since overflow from M'^j implies reference to M^r which in turn implies overflow from M^j also.

Case 2 : overflow from M^j, no overflow from M'^j :

- Suppose the last element in V_t' is not an element of V_t. Then starting from the end of V_t', if we eliminate those elements not in V_t, the two stacks will be in the same reverse ordering. This follows from LEMMA 8.3 and is illustrated in Figure 4.11. Thus we see that overflow from M^j, i.e., overflowing the last page of V_t, will not violate (a) since this page is still in V_t'. (b) is still preserved since the last page in V_t' is still not in V_t.
- Suppose V_t' and V_t are in the same reverse ordering. Then overflowing the last page of V_t does not violate (a) and results in the last page of V_t' not in V_t.

Case 3 : overflow from M^j and overflow from M'^j :

- Suppose the last element in V_t' is not in V_t. Referring to Figure 4.11 in Case 2, we see the result of overflowing the last element of V_t' and the last element of V_t does not violate (a) and still preserves the condition that the last element of V_t' is not in V_t
- Suppose V_t' and V_t are in the same reverse ordering. Then overflowing the last elements of V_t'

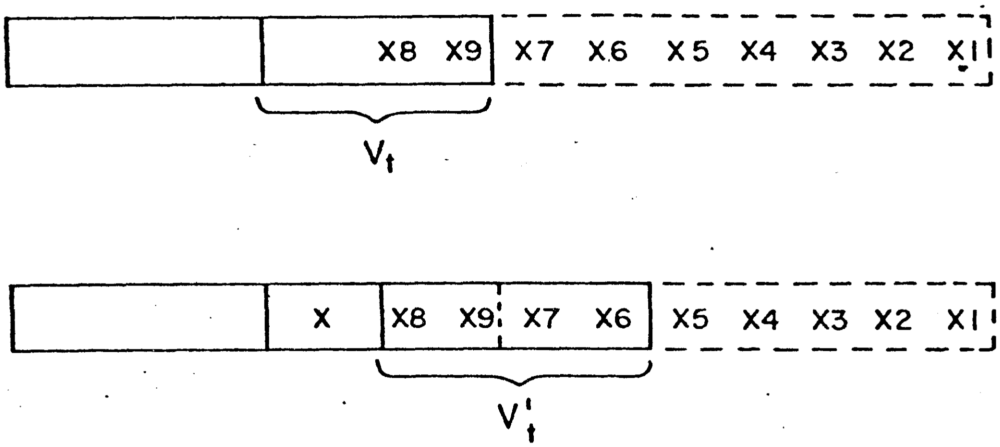


Figure 4.11

and V_t leaves V'_t and V_t still in the same reverse ordering. (a) is not violated since the same page is overflowed form M'^j and M^j.

Q.E.D.

PROOF of THEOREM 8

$M'^i \supseteq M^i$ for the same reasons as those used in THEOREM 7.

From LEMMA 8.4 $M'^j \supseteq M^j$.

Hence, $(M_t^i \cup (M_t^j)^i) \subseteq (M'^i_t \cup (M'^j_t)^i)$

Q.E.D.

4.5 SUMMARY

We have developed a formal model of a data storage hierarchy system specifically designed for very large databases. This data storage hierarchy makes use of different page sizes across different storage levels and maintains multiple copies of the same information in different storage levels of the hierarchy.

Four classes of algorithms obtained from natural extensions to the LRU algorithm are formally defined and studied in detail. Key properties of data storage hierarchy systems that make use of these algorithms are identified and formally proved.

It is found that for the LOCAL-LRU algorithms, no choice of sizes for the storage levels can guarantee that a lower storage level always contains all the information in the higher storage levels. For the GLOBAL-LRU algorithms, by choosing appropriate sizes for the storage levels, we can (1) ensure the above multi-level inclusion property to hold at all times for any reference string, (2) guarantee that no extra page references to lower storage levels are generated as a result of handling overflows, and (3) guarantee that no multi-level paging anomaly can exist.

Chapter V

DESIGN OF THE DSH-11 DATA STORAGE HIERARCHY SYSTEM

5.1 INTRODUCTION

In chapter 3, DSH-1, a general structure of the INFOPLEX data storage hierarchy system is introduced. DSH-1 incorporates novel features to enhance its reliability and performance. Many alternative architectures of data storage hierarchies can be derived from DSH-1. These structures can be used to perform detail studies of various design issues concerning data storage hierarchies.

This chapter describes a simple structure of the INFOPLEX data storage hierarchy derived from DSH-1. This structure is called DSH-11 and is used to develop detail protocols for supporting the read and write operations. Multi-level inclusion properties of DSH-11 are then discussed.

5.2 STRUCTURE OF DSH-11

The general structure of DSH-11 is illustrated in Figure 5.1. There are h storage levels in DSH-11, denoted by L(1), L(2), ... , L(h). L(1) is the highest performance storage level. L(1) consists of k memory ports. Each memory port

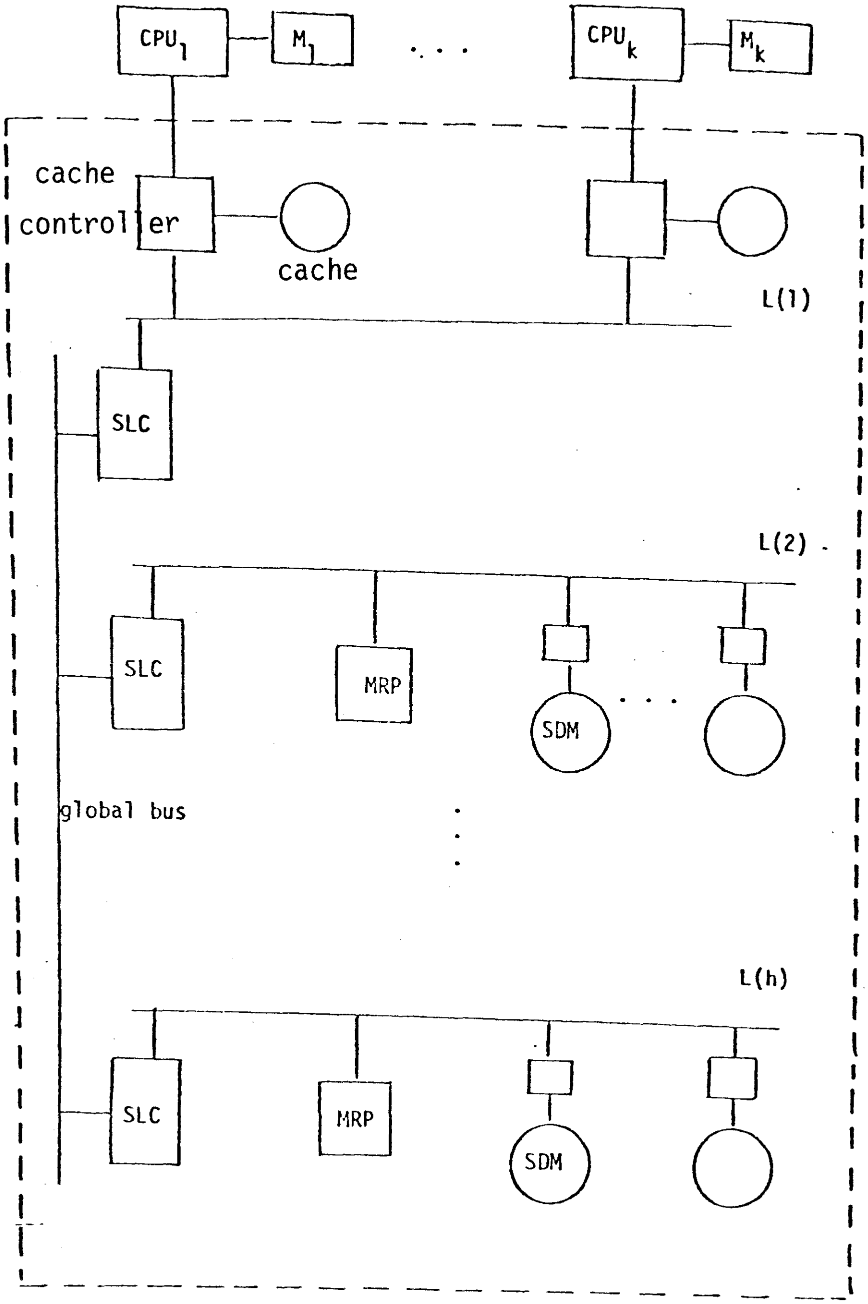


Figure 5.1 Structure of DSH-11

is connected to a processor. All k memory ports share the same local bus. A storage level controller (SLC) couples the local bus to the global bus. The global bus connects all the storage levels.

All other storage levels have the same basic structure. In each storage level, there is an SLC which couples the local bus to the global bus. There is a memory request processor (MRP) that handles requests to the storage level. There are a number of storage device modules (SDM's) that store the data within the storage level. The SLC, MRP, and SDM's share the same local bus.

The number of SDM's in different storage levels may be different. The type of storage device in the SDM's in different storage levels are different. For high efficiency, the block sizes of different storage levels are different. L(1) has a block size of q(1), L(2) has a block size of q(2)=n(1)*q(1), and so on, where n(1), n(2), ..., n(h-1) are non-zero integers.

5.2.1 The DSH-11 Interface

From the point of view of a processor connected to a DSH-11 memory port, DSH-11 appears as a very large virtual memory with 2**V bytes. The entire virtual address space is byte addressable. The instructions for a processor are stored in a local memory outside of DSH-11. This local memory has an address space of 2**G bytes. Hence, the effective data address space is (2**V-2**G) bytes.

All operations on data within DSH-11 are performed in L(1). Thus, if a referenced data item is not in L(1), it has to be brought into L(1) from a lower storage level. This induces a delay on the instruction comparable to a page fault in virtual memory systems. Each processor has multiple register sets to support efficient multiprogramming. The key registers for interfacing with DSH-11 are the memory operation register (MOR), the memory address register (MAR), the memory buffer register (MBR), the operation status register (OSR), and the process identification register (PIR). The MAR is V bits wide and the MBR is n bytes wide, where n is less than q(1), the block size of L(1).

A read operation requests n bytes of data at location pointed to by MAR to be brought into the MBR. A write operation requests the n bytes of data in the MBR be written to the location pointed to by the MAR. We shall assume that the n bytes of data in a memory reference do not cross a L(1) block boundary (If a data item crosses block boundaries, multiple requests will be used so that each request only reference data within block boundaries).

A _read_ or _write_ operation may proceed at the speed of the L(1) devices when the referenced data is found in L(1). Otherwise, the operation is interrupted and the processor switches to another process while DSH-11 moves the referenced data into L(1) from a lower storage level. When the data is copied into L(1), the processor is again interrupted to complete the operation.

5.2.2 The Highest Performance Storage Level - L(1)

As illustrated in Figure 5.1, L(1) consists of k memory ports. Each memory port consists of a data cache controller (DCC) and a data cache duplex (DCD). A DCC communicates with the processor connecting to the memory port. A DCC also maintains a directory of all data in the DCD. All k memory ports share a local bus. The SLC serves as a gateway for communication between L(1) and other storage levels.

5.2.3 A Typical Storage Level - L(i)

A typical storage level, L(1), consists of a number of SDM's, an MRP, and an SLC. An SLC serves as the gateway for communication among storage levels. The MRP services requests to L(i). An SDM performs the actual reading and writing of data. An SDM consists of a device controller and the actual storage device.

To gain high throughput, communications over the global bus is in standard size packets. The packet size is such that it is sufficient for sending one L(1) data block. Communications over a local bus at L(i) is also in standard size packets. The size of a packet depends on the storage level and is chosen so that a packet is sufficient to send a data block of size q(i).

In the following sections, the read and write operations are discussed in detail.

5.3 ALGORITHMS FOR SUPPORTING THE READ OPERATION

5.3.1 The Read-Through Operation

A read request is issued by a processor to its data cache controller. The data cache controller checks its directory to see if the requested data is in the data cache. If the data is found in the data cache, it is retrieved and returned to the processor. If the data is not in the data cache, a read-through request is queued to be sent to the next lower storage level, L(2), via the storage level controller.

At a storage level, a read-through request is handled by the memory request processor. The memory request processor checks its directory to determine if the requested data is in one of the storage device modules at that level. If the data is not in the storage level, the read-through request is queued to be sent to the next lower storage level via the storage level controller.

If the data is found in a storage level, L(i), a block containing the data is retrieved by the appropriate storage device module and passed to the storage level controller. The storage level controller broadcasts the block to all upper storage levels in several standard size packets. Each upper storage level has a buffer to receive these packets. A storage level only collect those packets that assemble into a sub-block of the appropriate size that contains the requested data. This sub-block is then stored in a storage device module.

At L(1), the sub-block containing the requested data is stored, and the requested data is sent to the processor with the proper identification.

Figure 5.2 illustrates the read-through operation. Assume that DSH-11 has only three storage levels, L(1) with block size b, L(2) with block size 2b, and L(3) with block size 4b. Suppose a reference to a data item 'x' is found in a block in L(3). Then the sub-block of size 2b containing 'x' is broadcasted to L(2) and L(1) simultaneously. L(2) will accept and store the entire sub-block of size 2b. L(1) will only accept and store a block of size b that contains

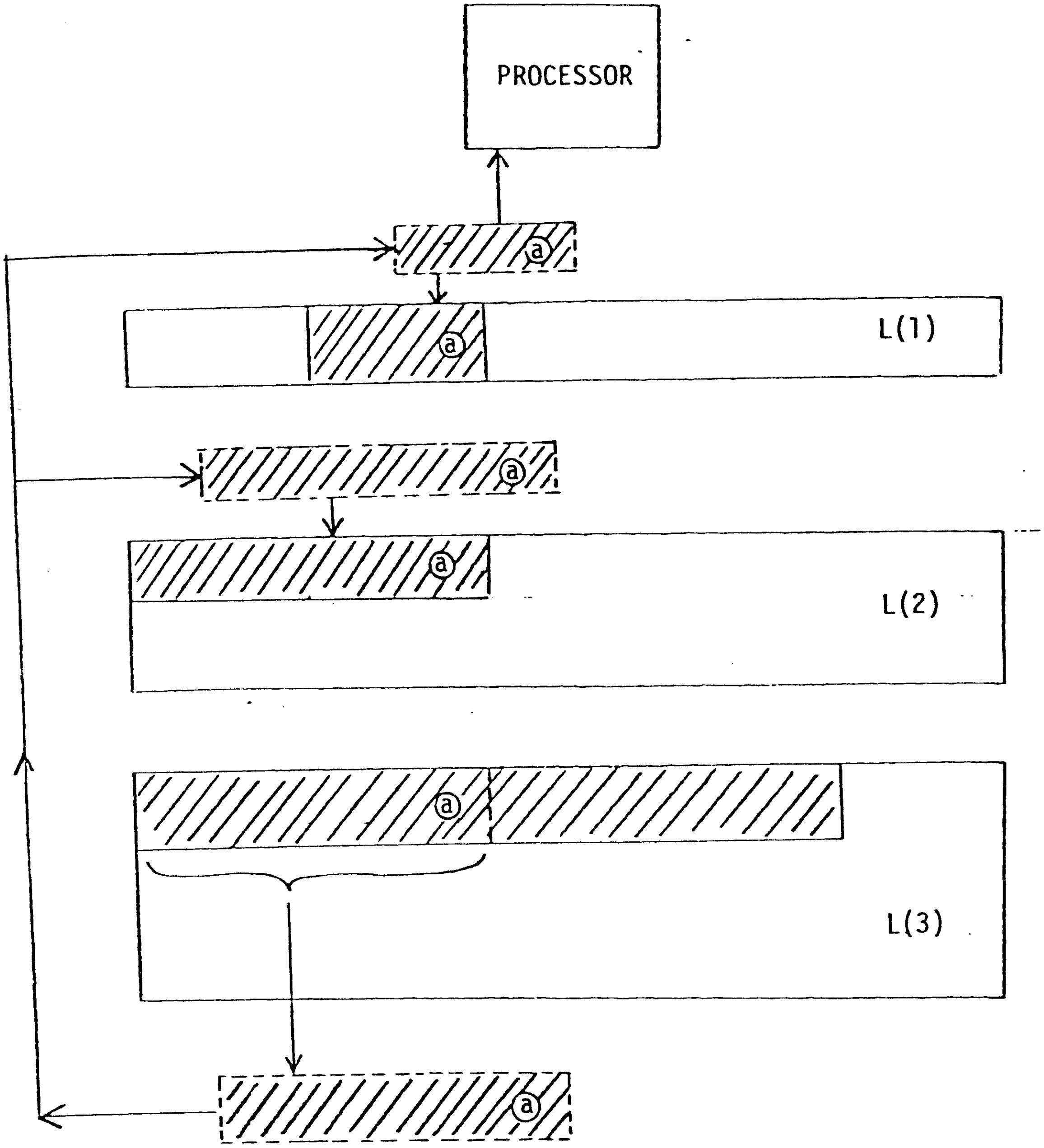


Figure 5.2 Illustration of Read Through

'x'. The two sub-blocks, each of size 2b of a parent block in L(3) are referred to as the child blocks of the parent block.

5.3.2 Overflow Handling

To accomodate a new data block coming into a storage level as a result of a read-through, an existing data block may have to be evicted. The block chosen for eviction is that which is the least recently referenced block such that none of its child blocks is in the immediate upper storage level. To support this scheme, each block is associated with an Upper Storage Copy Code (USC-code). If any of the child blocks of a data block is in the immediate upper storage level, its USC-code is set. Each time the last child block in L(i) of a parent block in L(i+1) is evicted from L(i), an overflow request is sent to L(i+1) to reset the USC-code of the parent block.

Blocks in L(1) and L(2) are handled slightly differently due to the use of data caches in L(1). Since multiple copies of the same data block may be in different data caches, evicting a block does not necessarily guarantee no other copy exists in L(1). The following strategy is adopted to overcome this difficulty. During a read-through, the USC-code of the block in L(2) is incremented by 1. Each time a block in L(1) is evicted, an overflow request is sent to L(2) to decrement the USC-code of the corresponding parent block. This strategy does not require communications among different data caches.

5.3.3 Pathological Cases of Read-Through

The parallel and asynchronous operations of DSH-11 and the use of buffers at each storage level complicates algorithms for handling the read operation. Pathological cases that affect the algorithms are discussed below.

5.3.3.1 Racing Requests

Two different requests R1 and R2 may reference the same block of data. Furthermore, these two requests may be closed to each other such that both may be reading-through the same block of data at some storage level. Since a data block is transmitted in several packets asynchronously, each packet must be appropriately identified to avoid confusion when assembling the data sub-blocks at higher storage levels.

A similar situation arises when R1 and R2 are closed together such that R2 begins to read-through the same data block that has just been read-through by R1. Thus a data block arriving at a storage level may find that a copy of it already exists at that storage level. In this case, the incoming data block is ignored. At L(1), this data block is ignored and the one in the data cache is read and returned to the processor, since this is the most recent copy of the data block.

5.3.3.2 Erronous Overflow

When a data block is evicted from L(i) to make room for an incoming data block, an overflow request containing the virtual address of the evicted data block may be generated. The purpose of the overflow request is to inform L(i+1) that there is no longer any data block in L(i) with the same family address as the virtual address in the overflow request. Hence, an overflow request is generated only when the last member of a family in L(i) is evicted.

The overflow request has to be forwarded to the MRP at L(i+1). At any point on the way to the MRP, a data block in the same family as the evicted block may be read-through into L(i). This poses the danger that when the MRP receives the overflow request indicating that no data block in the same family as the evicted block exists in L(i), there is actually one such block being placed in L(i).

The following strategy is incorporated in the algorithms that support the read-through operation to avoid such a potential hazard.

1. At the time the *overflow* request is to be created in L(i), a check is made to see if there is any data block in the same family as the evicted block that is currently in any buffer of L(i) waiting to be placed in L(i). If so, the *overflow* request is not created.

2. At the time a new data block arrives in L(i), any *overflow* request with the same family address as the incoming data block waiting to be sent to L(i+1) is purged.

3. When an *overflow* request arrives at L(i+1) from L(i), a check is made to see if there is any data block waiting to be sent to L(i) that has the same family address as the *overflow* request. If so, the *overflow* request is purged.

4. At the time a request is generated to send a data block to L(i), any *overflow* request from L(i) that is still waiting in L(i+1) that has the same family address as the data block to be sent to L(i) is purged.

5.3.3.3 Overflow to a Partially-assembled Block

Suppose that as a result of a read-through from L(i+2), B(i), the only child block of B(i+1), is in L(i) and B(i+1) is partly assembled in the buffer in L(i+1). It is possible that B(i+1) is still partly assembled in L(i+1) when B(i) is evicted from L(i). The *overflow* request will find that the corresponding parent data block is still being assembled in L(i+1). A solution to this difficulty is to check, at the time of arrival of the *overflow* request, if there is any incoming data block which is the target parent block of the evicted block as indicated in the *overflow* request. If so, the *overflow* request is held till this parent block has been placed in a storage device.

5.3.4 Transactions to Handle the Read Operation

A read request is issued by a processor to its data cache controller. If the data is not found in the data cache, it has to be brought up via a read-through. The read-through operation is realized via a number of transactions. The flow of transactions to support the read-through operation is illustrated in Figure 5.3.

A read-through transaction is created by a data cache controller and propagated to lower storage levels. At each storage level, the read-through transaction is handled by a memory request processor which checks its directory to see if the requested data is in the current storage level. If the data is not in the current storage level, the read-through transaction is sent to the next lower storage level.

Suppose that the data requested is found at L(i). The read-through transaction is terminated and a retrieve transaction is created to read the data from a storage device. A read-response-out transaction that contains the read data is sent to the storage level controller. The storage level controller generates a number of read-response-packet transactions which are broadcasted to all higher storage levels. Each of these transactions contains a sub-block of the requested data.

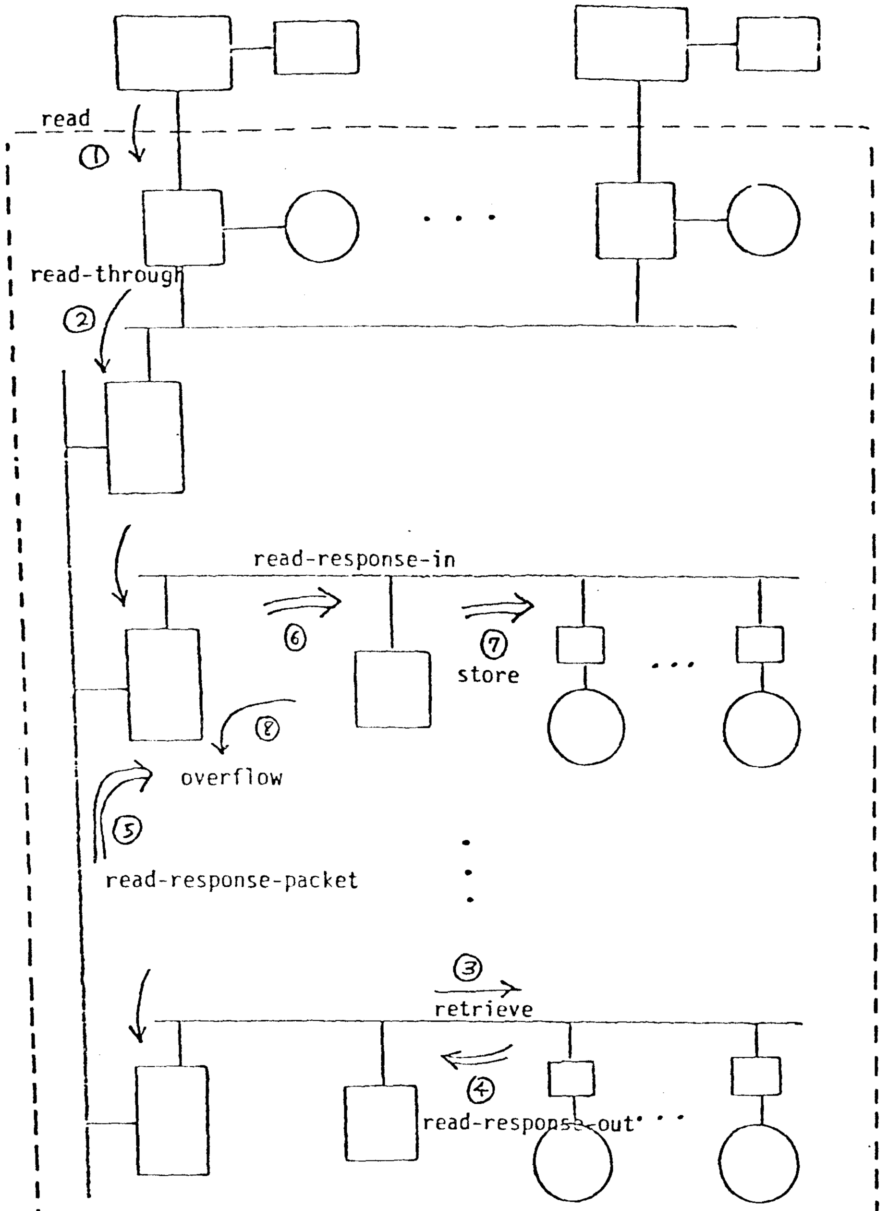


Figure 5.3 Transactions to Support Read Through

At each higher storage level, an appropriate number of read-response-packet transactions are assembled into a read-response-in transaction which contains a data block of the appropriate size. The memory request processor obtains free space for the new data block in the read-response-in transaction either by using existing free space or by evicting an existing block. Eviction of an existing data block may result in an overflow transaction being sent to the next lower storage level. At the memory request processor, the read-response-in transaction is serviced and a store transaction is created. A storage device module handles the store transaction by writing the data to a storage device.

The following subsections describe the algorithms for handling each of the above transactions.

5.3.4.1 The read-through Transaction

The read-through transaction is created by a data cache controller and propagated down the storage levels via the storage level controllers. It has the following format:
(read-through, virtual-address, process-id),
where virtual-address is the virtual address of the referenced data item, and process-id consists of a CPU identifier and a process number. It is the identifier of the process that generated the read operation. The transaction is han-

dled by a memory request processor using the following algorithm.

1. Search directory for read-through.virtual-address.

2. If not found, forward the transaction to the storage level controller, which will send it to the next lower storage level.

3. If found, suppose it is in the i-th directory entry, and suppose directory(i).TRANSIT-code is not set, do:

   i) Set directory(i).USC-code to indicate a child block exists in the higher storage level for this block. If this is level L(2), increment directory(i).USC-code instead of setting it.

   ii) Set directory(i).HOLD-code to forbid any overflow to this block while the data is being retrieved.

   iii) Create a retrieve transaction :(retrieve, virtual-address,directory(i).real-address,process-id).

   iv) Send the retrieve transaction to the appropriate storage device module.

4. If found, suppose it is in the i-th directory entry, and suppose directory(i).TRANSIT-code is set, then hold the request and retry later. When the TRANSIT-code is set, it indicates that the corresponding data block is in transit, hence any reference to it is not allowed.

5. End.

5.3.4.2 The retrieve Transaction

The retrieve transaction is created by a memory request processor and handled by a storage device module as follows.

1. Read the data block using retrieve.real-address.

2. Create a read-response-out transaction :(read-response-out, virtual-address, process-id, data), where data is the block containing the referenced data item.

3. Send the read-response-out transaction to the storage level controller.

4. End.

5.3.4.3 The read-response-out Transaction

The read-response-out transaction is created by a storage device module and handled by a storage level controller using the following algorithm.

1. Purge any incoming overflow transaction that has the same family address as read-response-out.virtual-address.

2. Send (update-directory, virtual-address,HOLD-code=Ø) to memory request processor to reset the HOLD-code, so that overflow to this block is now allowed.

3. Broadcast the transaction (reserve-space, virtual-address, process-id) to all higher storage levels to reserve buffer space for assembling read-response-out.data.

4. Wait till all higer levels have acknowleged the space reservation transaction.

5. Generate the appropriate number of (read-response-packet, virtual-address, process-id, data, data-virtual-address) transactions. Data is a standard size sub-block and data-virtual-address is the virtual address of this sub-block.

6. Broadcast each read-response-packet transaction to all higher storage levels.

7. End.

5.3.4.4 The read-response-packet Transaction

This transaction is created by a storage level controller and broadcasted to all higher storage level controllers where they are assembled into read-response-in transactions to be handled by the memory request processors. Note that a storage level only accepts those packets relevant for assembling into a data block of the appropriate size, all other associated packets are ignored. The following algorithm is used by a storage level controller in assembling the read-response-in transactions.

1. If this is the first packet of the assembly, do:

   i) Purge any outgoing overflow transaction that has the same family address as the block being assembled.

   ii) Add the packet to the assembly.

2. If this is an intermediary packet of the assembly, simply add it to the assembly.

3. If this is the last packet of the assembly, do:

   i) Replace the assembly by a (read-response-in, virtual-address, process-id, data) transaction. Data is the block just assembled.

   ii) Send the above transaction to the memory request processor.

4. End.

5.3.4.5 The read-response-in Transaction

This transaction is created by a storage level controller and sent to a data cache controller (for L(1)) or to a memory request processor (for L(2), L(3), ...).

The following algorithm is used by a data cache controller in handling this transaction.

1. Purge any outgoing <u>overflow</u> transaction that has the same family address as the block in the <u>read-response-in</u> transaction.

2. Search directory for read-response-in.virtual-address.

3. If found, suppose it is the i-th directory entry, do:

   i) Read data from the data cache using directory(i).real-address.

   ii) Send data to the processor.

   iii) Increment directory(i).USC-code by 1.

4. If not found, do:

   i) Select a block to be evicted (assuming that data cache is full). This is the least recently referenced block such that it is not engaged in a stored-behind process. Suppose this block corresponds to directory(i).

   ii) Obtain directory(i).virtual-address, directory(i).USC-code, and directory(i).real-address.

   iii) Write read-response-in.data into location directory(i).real-address in the data cache.

   iv) Return read-response-in.data to the processor.

v) Create (overflow, directory(i).virtual-address, USC-code=directory(i).USC-code) transaction, send it to the storage level controller.

vi) Update directory(i).virtual-address with read-response-in.virtual-address.

vii) Set directory(i).USC-code to 1.

5. End.

At a memory request processor, the read-response-in transaction is handled as follows.

1. Purge any outgoing overflow transaction with the same family address as the data block in the read-response-in transaction.

2. Search for read-response-in.virtual-address in the directory.

3. If not found, do:

    i) Select a block to be evicted (assuming that the storage level is full). This is the least recently referenced block such that it is not engaged in a store-behind process, it is not held (i.e., HOLD-code = 0), and it is not in transit (i.e., TRANSIT-code = 0). Suppose this block corresponds to directory(i).

    ii) Obtain directory(i).virtual-address and directory(i).real-address.

    iii) If the evicted block is the last of its family in the storage level and that there is no incoming block with the same family address then create a (overflow, directory(i).virtual-address, USC-code=1) transaction. Send the transaction to the storage level controller to be sent to the next lower storage level.

    iv) Set directory(i).TRANSIT-code to 1 to indicate the corresponding block is in transit.

v) Update directory(i).virtual-address with read-response-in.virtual-address.

vi) Set directory(i).USC-code to 1.

vii) Create a (store, directory(i).real-address, data) transaction and send it to the appropriate storage device module.

4. End.

5.3.4.6 The store Transaction

This transaction is handled by a SDM. Store.data is placed in store.location, and a transaction (update-directory, virtual-address, TRANSIT-code = Ø) is sent to the MRP to reset the TRANSIT-code so that references to this block is now allowed.

5.3.4.7 The overflow Transaction

This transaction is created by a data cache controller or a memory request processor and routed to a memory request processor in the lower storage level via the storage level controllers. At each stop on the way to a memory request processor, a check is made to see if any incoming data block has the same family address as the overflow transaction. If so, the following algorithm is executed.

1. If the direction of flow of the overflow and read-response-in are opposite, the overflow is purged.

2. If the direction of flow of the overflow and the read-response-out are opposite, the overflow is purged.

3. If the two transactions are in the same direction of flow, the overflow is held to be processed after the read-response-in is handled.

At a memory request processor, if the HOLD-code is set for the parent block of the overflowed block, the overflow transaction is purged (HOLD-code is set indicates that the block is being retrieved by an SDM to be read-through to all upper storage levels). Otherwise, the USC-code of the parent block is decremented by overflow.USC-code.

5.4 ALGORITHMS TO SUPPORT THE WRITE OPERATION

Algorithms to support the write operation are simplified by the multi-level inclusion properties of DSH-11. The multi-level inclusion properties of DSH-11 guarantee that all the data items in L(i) is contained in L(i+1). Thus, when writing a child block in L(i) to its parent block in L(i+1), the parent block is guaranteed to exist in L(i+1). The multi-level inclusion properties of DSH-11 will be discussed in a later section.

5.4.1 The Store-Behind Operation

After a block is placed in a data cache as a result of a read-through operation, its parent block exists in L(2), and its grand-parent block exists in L(3), and so on. Due to the multi-level inclusion properties of DSH-11, this situation will persist as long as the block is in the data cache.

After a data block in a data cache is updated, it is sent down to the next lower storage level to replace the corresponding child block in its parent block. This updated parent block is sent down to the next lower storage level to update its parent block, and so on. This process is refered to as the store-behind operation and takes place at slack periods of system operation.

DSH-11 uses a two-level store-behind strategy. This strategy ensures that an updated block will not be considered for eviction from a storage level until its parent and grand-parent blocks are updated. This scheme will ensure that at least two copies of the updated data exists in DSH-11 at any time. To support this scheme, a Store-Behind Code (SB-code) is associated with each data block in a storage level. The SB-code indicates the number of acknowledgements from lower storage levels that the block must receive before it can be considered for eviction.

In a write operation, the data item is written into the data cache duplex, and the processor is notified of the completion of the write operation. we shall assume that the data item to be written is already in L(1) (This can be realized by reading the data item into L(1) before the write operation). A store-behind operation is next generated by the data cache controller and sent to the next lower storage

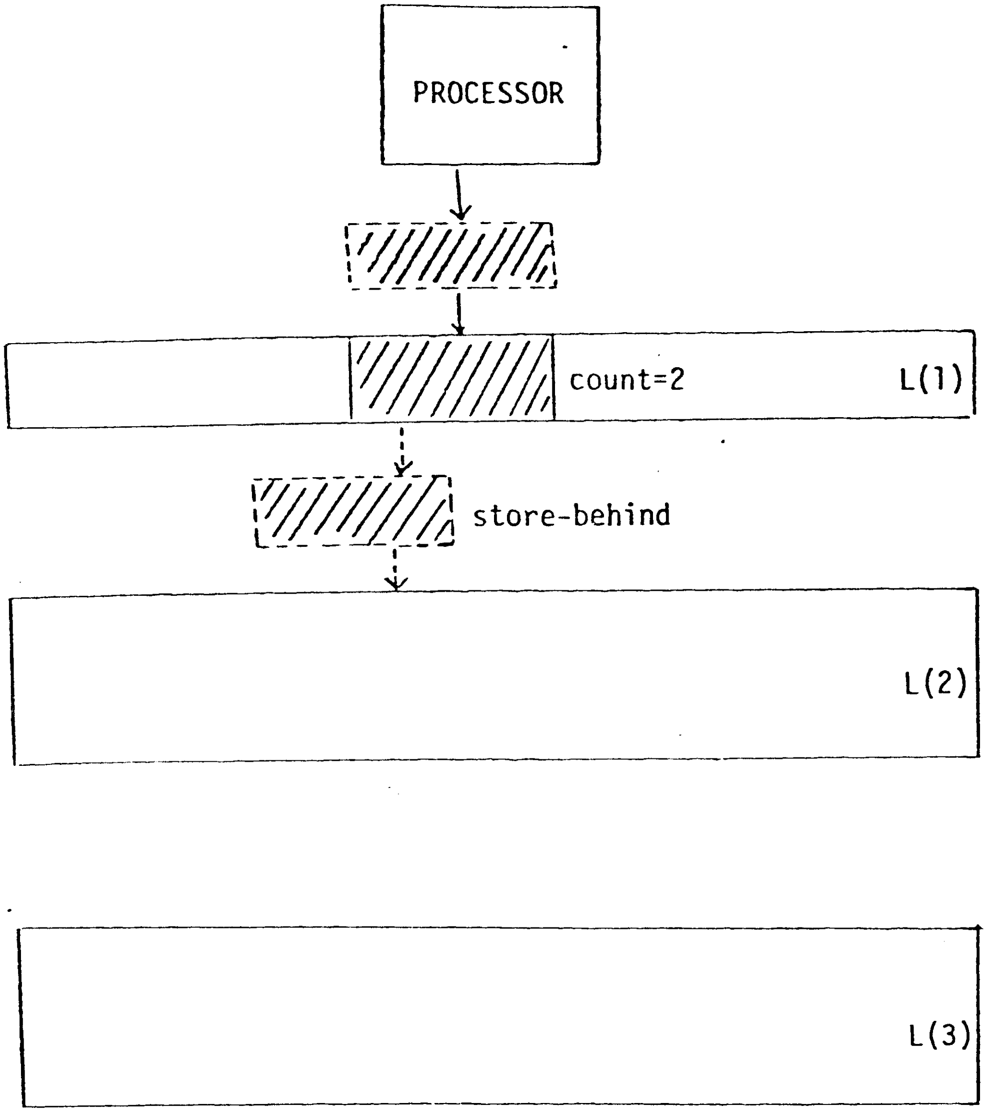


Figure 5.4(a) Illustration of Store Behind (a)

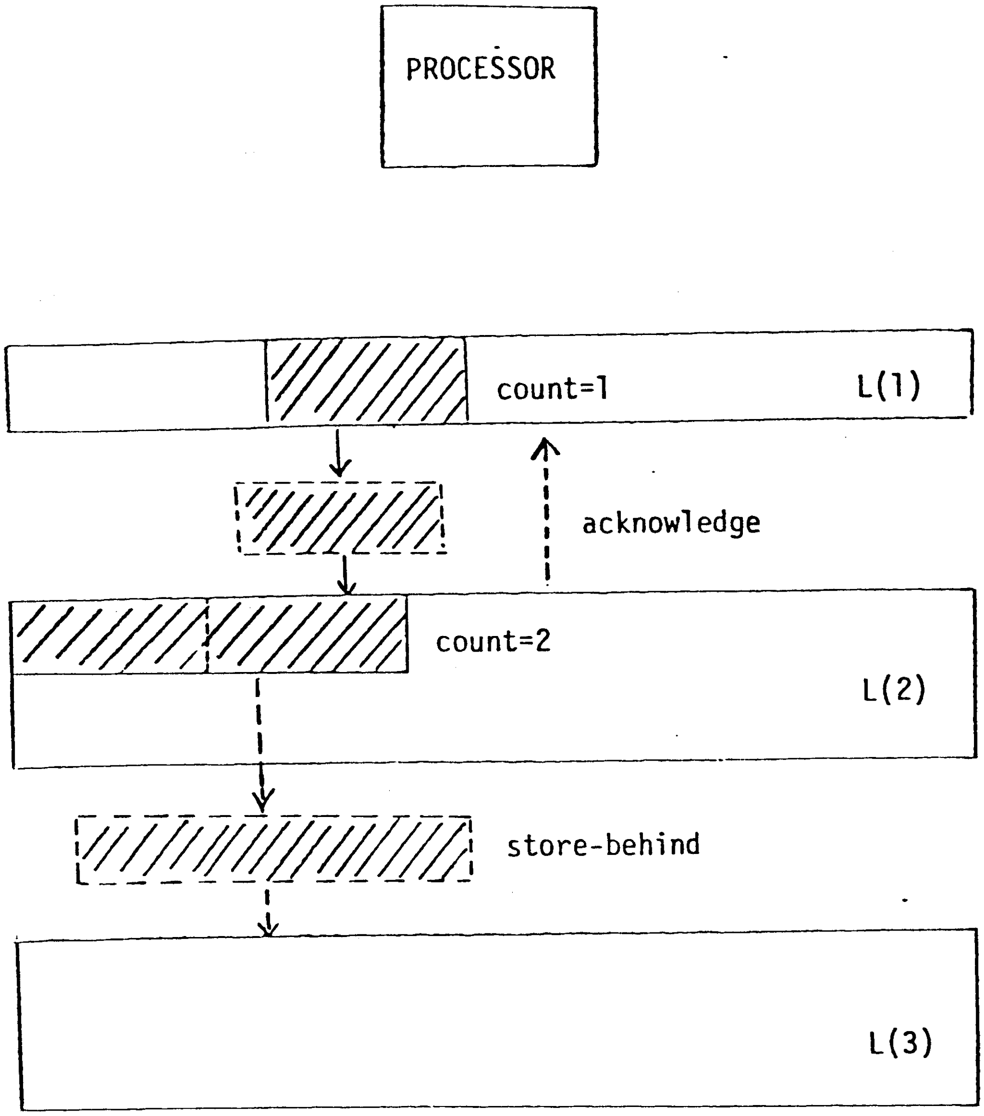


Figure 5.4(b) Illustration of Store Behind (b)

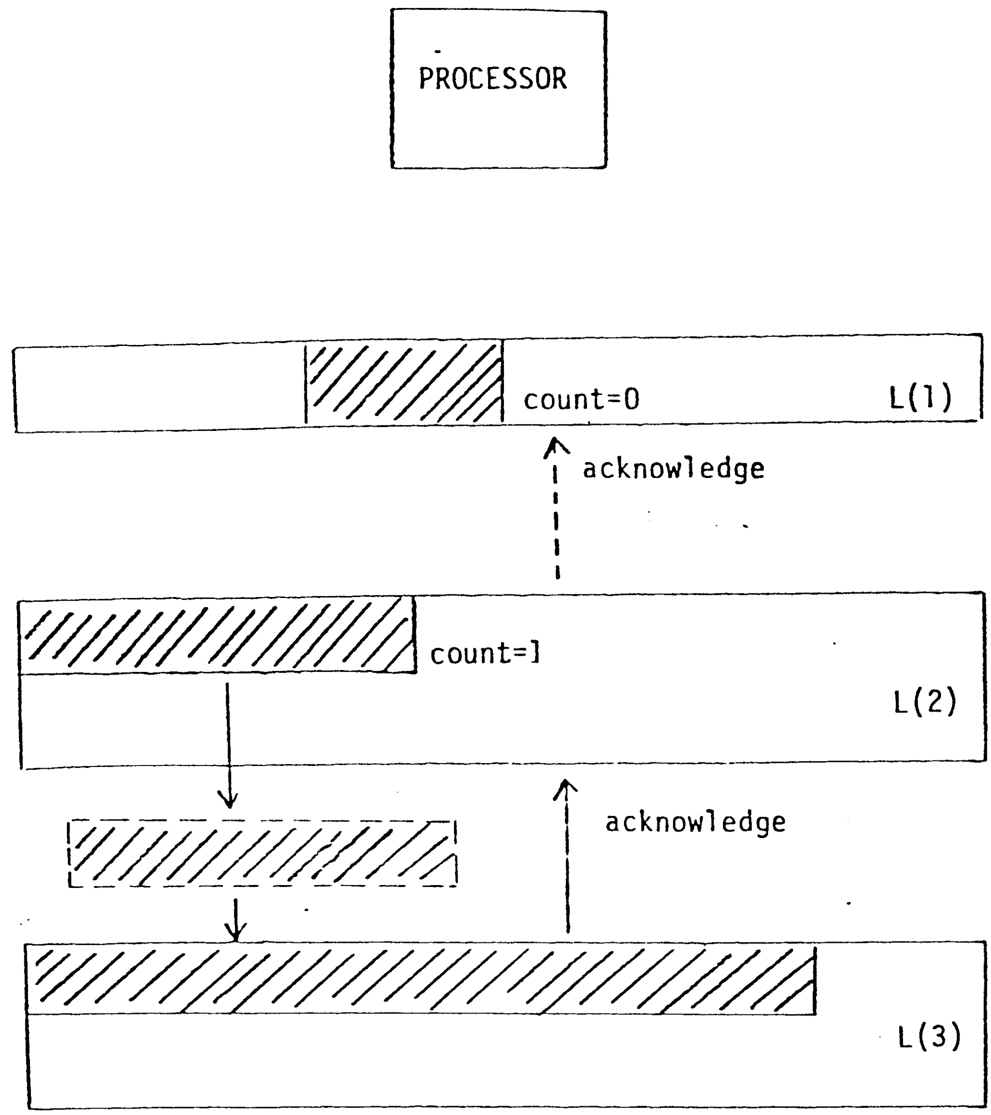


Figure 5.4(c) Illustration of Store Behind (c)

level. The block in L(1) that has just been updated is marked with a count of 2. This is illustrated in Figure 5.4(a).

When a store-behind operation is received in L(2), the addressed data is written, and marked with a count of 2. An acknowledgement is sent to the next upper storage level, L(1), and a store-behind operation is sent to the next lower storage level, L(3). When an acknowledgement is received at L(1), the counter for the addressed data item is decremented by 1, which becomes 1. This is illustrated in Figure 5.4(b).

The store-behind is handled in L(3) by updating the appropriate data block. An acknowledgement is sent to L(2). At L(2), the corresponding block counter is decremented by 1, which becomes 1. The acknowledgement is forwarded to L(1). At L(1), the corresponding block counter is decremented by 1 which now becomes Ø, hence the block is elligible for replacement. This is illustrated in Figure 5.4(c).

Thus we see that the two-level store-behind strategy maintains at least two copies of the written data at all times. Furthermore, lower storage levels are updated at slack periods of system operation, thus enhancing performance. Detail algorithms for supporting this scheme will be discussed in a later section.

5.4.2 Lost Updates

Several different updates to the same block will result in several different store-behind requests be sent to the next lower storage level. It is possible that these store-behind requests arrive at the next storage level out of sequence, resulting in lost updates.

To resolve this potential hazard, there is a time-stamp associated with each block indicating the last time the block was updated. There is also a time-stamp associated with each child block of the parent block indicating the last time the child block was updated by a store-behind operation. A store-behind request will contain the block to be updated and its time-stamp. This time-stamp will be compared with that of the corresponding child block in the target parent block. Only when the store-behind data is more recent will the update to the target parent block be performed.

5.4.3 Transactions to Support the Write Operation

Figure 5.5 illustrates the transactions to support the write operation. We shall assume that the target block of a write operation already exists in a data cache. This can be ensured by first reading the target block before issuing the write request to the data cache. After the data is written into a target data block in a data cache, a store-behind

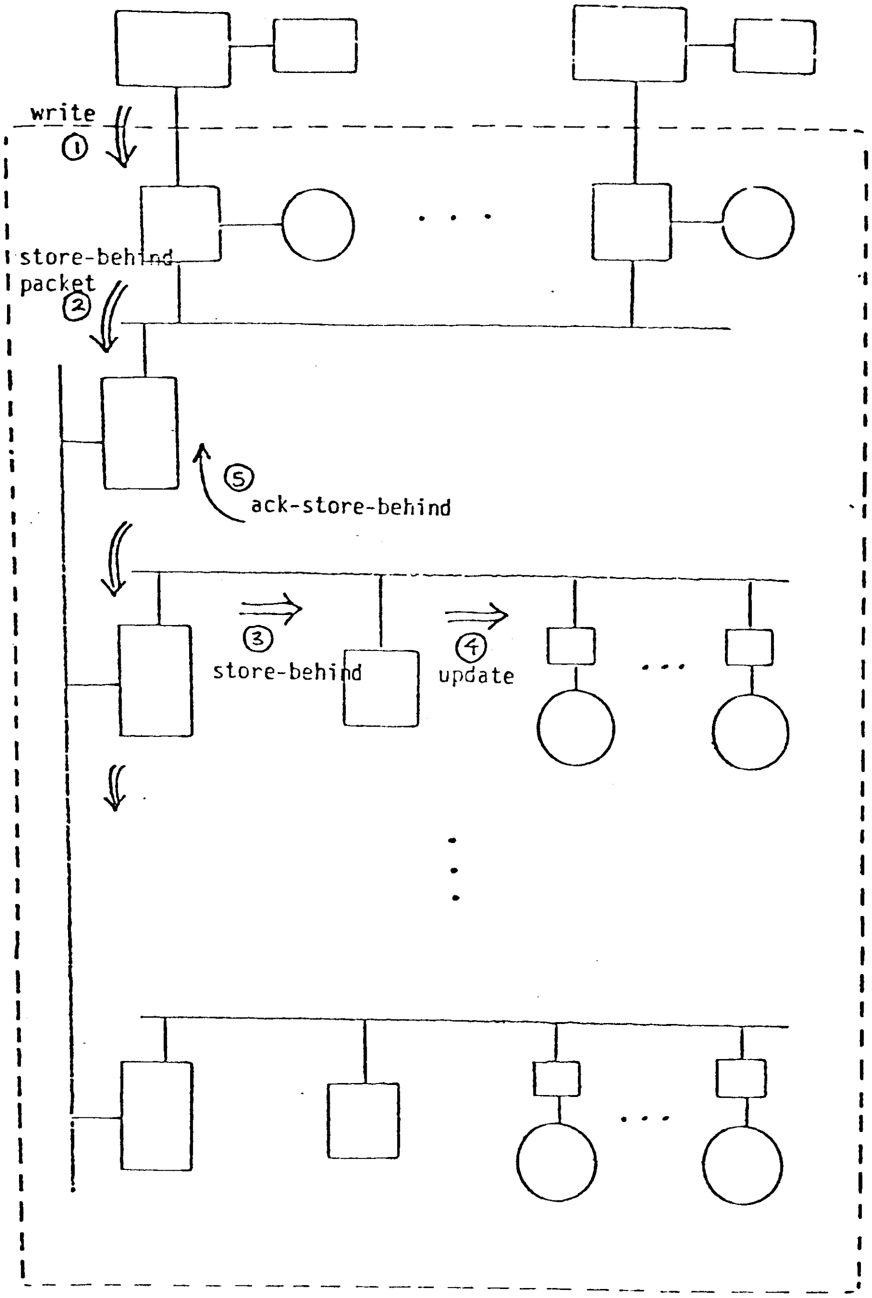


Figure 5.5 Transactions to Support Store Behind

transaction containing the updated block is sent to the next lower storage level. The store-behind transaction is serviced by the memory request processor. The memory request processor generates an update transaction and sends it to the appropriate storage device module. The memory request processor also sends an ack-store-behind transaction to the higher storage level. The storage device module handles the update transaction by replacing the corresponding child block in the target parent block with the data in the store-behind transaction. Another store-behind transaction containing the updated parent block is created and sent to the storage level controller to be forwarded to the next lower storage level.

A store-behind transaction is sent to the next lower storage level in several standard size packets, each corresponds to a store-behind-packet transaction. At a storage level controller, these packets are assembled into the original store-behind transaction. The algorithms for sending and assembling packets are very similar to those used for the read-through operation and will not be repeated here. The following describes the algorithms for supporting the above transactions to realize the write operation.

5.4.3.1 The store-behind Transaction

A store-behind transaction has the following format:

(store-behind, virtual-address,process-id,data,time-stamp)

This transaction is handled by a memory request processor using the following algorithm.

1. Search directory for store-behind.virtual-address.

2. If not found, hold the transaction and retry after a time out, because the target parent block is still being assembled in the buffer.

3. If found, compare store-behind.time-stamp with the time-stamp of the corresponding child block of the target parent block.

4. If store-behind.data is more current than the child block, do:

    i) Send (update, virtual-address, data, real-address, time-stamp-of-parent) to the appropriate storage device module.

    ii) Update the time-stamp of the child block with store-behind.time-stamp.

    iii) Send (ack-store-behind, virtual-address, process-id, ACK-code = 2) to the immediate higher storage level. ACK-code indicates the number of levels this transaction is to be routed upwards.

5. If store-behind.data is not more current than data in storage level, send two (ack-store-behind, virtual-address, process-id, ACK-code = 2) to the immediate higher storage level.

6. End.

5.4.3.2 The update Transaction

The update transaction is handled by a storage device module using the following algorithm.

1. Replace the appropriate child block in the target parent block by update.data.

2. The updated target parent block is retrieved.

3. Send (update-directory, virtual-address, SB-code = 2) to the memory request processor to increment SB-code of the target parent block by 2.

4. (store-behind, virtual-address, process-id,target-parent-block, time-stamp = update.time-stamp-of-parent) is sent to the storage level controller to be sent to the next lower storge level.

5. End.

5.4.3.3 The ack-store-behind Transaction

This transaction is handled by a memroy request processor. The algorithm used is as follows.

1. The SB-code of the corresponding block is decremented by 1.

2. The ack-store-behind.ACK-code is decremented by 1.

3. If ack-store-behind.ACK-code is greater than Ø the forward the ack-store-behind to the immediate upper storage level.

4. End.

5.5 MULTI-LEVEL INCLUSION PROPERTIES

As a result of the read-through operation, the block read-through into L(1) leaves its 'shadow' in every lower storage level that participated in the read-through operation. Is it true then, that a storage level, L(i), always contains every data block in L(i-1)? When this is true, multi-level inclusion (MLI) is said to hold.

It has been formally proved in Chapter 4 that certain algorithms incorporating the read-through strategy can guarantee MLI provided that the relative sizes of the storage levels be appropriately chosen. Furthermore, it is found that certain other algorithms can never guarantee MLI. This section explores the MLI properties of DSH-11. In the following sections, the importance of MLI is briefly reviewed, a model of DSH-11 is developed, and the MLI property of DSH-11 is analyzed informally.

5.5.1 Importance of MLI

The MLI properties have important implications for the performance and availability of DSH-11. First, since the block size of L(i) is larger than that of L(i-1), L(i) can be viewed as an extension of the spatial-locality (Madnick, 1973) of L(i-1). Second, except for the lowest storage level, each data item has at least two copies in different storage levels. Hence, even the failure of an entire storage level will not result in data loss. Third, algorithms to support the write operation is simplified if MLI holds because a store-behind operation always finds the target parent data block exists in a storage level.

5.5.2 A Model of DSH-11

Figure 5.6 illustrates a model of DSH-11. DSH-11 has h storage levels, L(1), L(2), ... , L(h). L(1) consists of k data caches. Each data cache consists of a buffer B(1,i), and a storage, M(1,i). All the buffers of the data caches are collectively denoted as B(1), and all the storage of the data caches are collectively denoted as M(1). The size of B(1,i) is b(1,i) number of blocks of size q(1). The size of M(1,i) is m(1,i) number of blocks of size q(1). Hence L(1) has b(1) = b(1,1) + b(1,2) + ... + b(1,k) blocks of buffer space and m(1) = m(1,1) + m(1,2) + ... + m(1,k) blocks of storage space.

A buffer is for holding data blocks coming into or going out of the storage level. A data block may be partially assembled in a buffer. Only data blocks in the storage space are accounted for by the directory. Note that a buffer is not used for holding transactions that do not contain any data, e.g. an ack-store-behind transaction does not occupy any buffer space.

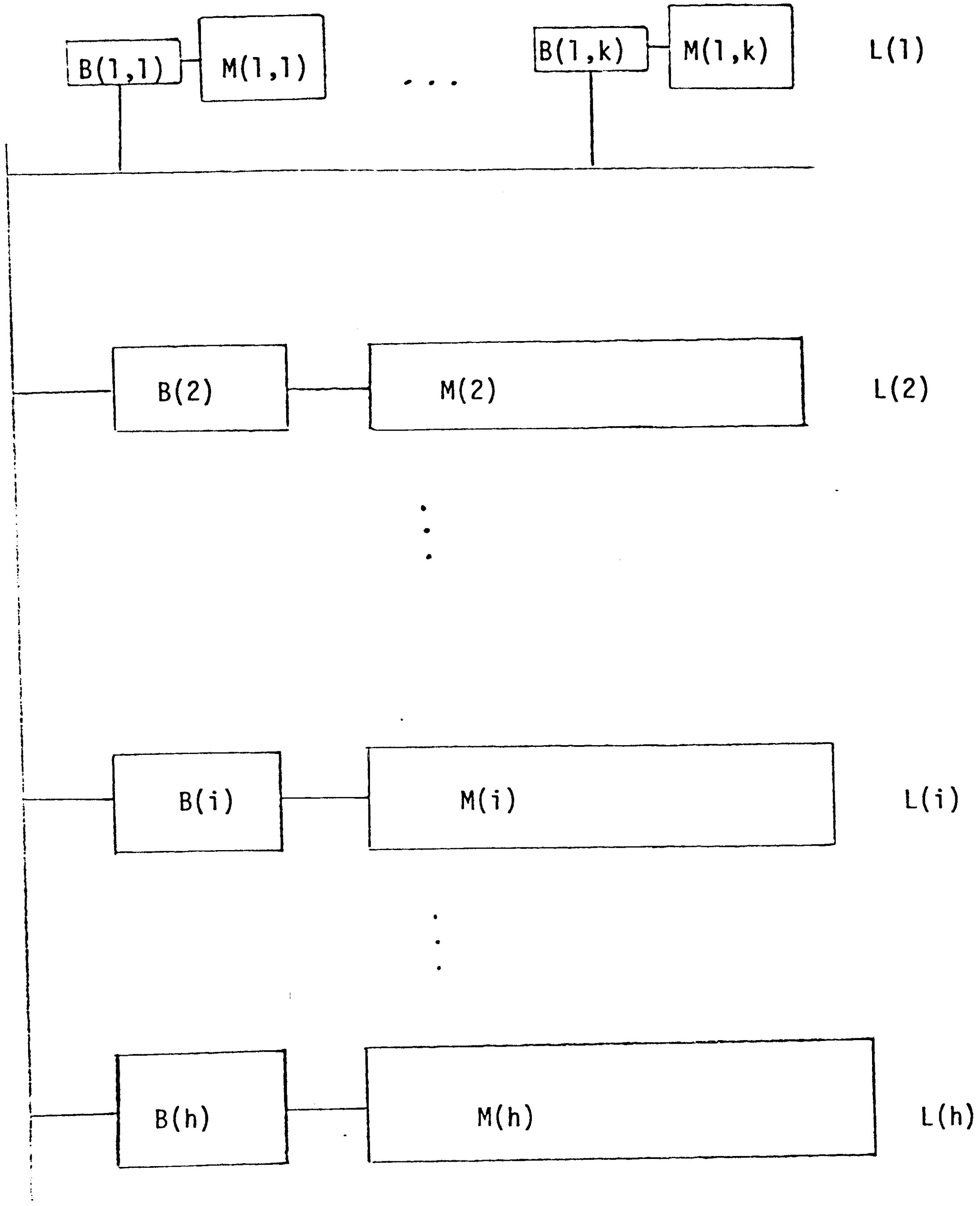


Figure 5.6 A Model of DSH-11

A typical storage level, L(i), consists of a buffer B(i), and a storage, M(i). The size of B(i) is b(i) number of blocks each of size q(i). The size of M(i) is m(i) number of blocks each of size q(i). The block size of L(i) is q(i), where q(i) = n(i-1)*q(i-1), for i = 2, 3, ... , h. The n(i)'s are integers.

5.5.3 MLI Properties of DSH-11

Based on the model in the previous section, the MLI condition can be stated as follows: a data block, whole or partially assembled, that is found in L(i) is also found in L(i+1). This section shows that for DSH-11, it is possible to guarantee the following: (1) MLI holds at all times, (2) it is always possible to find a block for eviction to make room for an incoming block, and (3) an overflow transaction from L(i) always finds its target parent block in L(i+1).

Proposition

Let J = i + 1. Using the algorithms described in the previous sections, if m(j) is greater than m(i)+b(i) then

1. MLI holds for L(i) and L(j), i.e., any block found in L(i) can be found in L(j),

2. If block replacement in M(j) is required, there is always a block not in L(i) that can be considered for overflow, and

3. An overflow transaction from L(i) always contains the address of a block that can be found in M(j).

Proof of Proposition

There are three cases:

1. There are no overflows from L(i): Since m(j) is greater than m(i)+b(i), no overflow from L(i) implies no overflow from L(j). Thus all blocks present in L(i) are still in L(j), i.e., (1) is true.

2. There are overflows from L(i), no overflow from L(j): No overflow from L(j) implies that all blocks referenced so far are still in L(j). Thus any block in L(i) is still in L(j), i.e., (1) is true. Since any overflow from L(i) will find the block still in L(j), (3) is true.

3. There are overflows from L(j): Consider the first overflow from L(j). Just before the overflow, (1) is true. Also just before the overflow, M(j) is full. M(j) is full and m(j) is greater than m(i)+b(i) implies that there is at least one block in M(j) that is not in L(i) (i.e., their USC-code = Ø). Choose from these blocks the least recently referenced block such that its SB-code = Ø. If no such block exists, wait, and retry later. Eventually the store-behind process for these blocks will be terminated and these blocks will be released. Thus a block will be available for overflow from M(j). Thus (2) is true. After the overflow, (1) is still preserved. (1) and (2) implies (3).

If next reference causes no overflow from L(j), then the arguement in Case 2 applies. If the next reference causes overflow from L(j), then the arguement in Case 3 applies.

5.6 SUMMARY

The DSH-11 design, a data storage hierarchy for the INFOPLEX data base computer, is described. Protocols for supporting the read and write operations in DSH-11 are described in detail. It is then shown that DSH-11 is able to guarantee multi-level inclusion at all times for any reference string provided that the sizes of the buffers and storage at the storage levels are chosen appropriately.

Chapter VI

SIMULATION STUDIES OF THE DSH-11 DATA STORAGE HIERARCHY SYSTEM

6.1 INTRODUCTION

This chapter discusses the results of a series of simulation studies of the DSH-11 data storage hierarchy system.

A key objective of these simulation studies is to assess the feasibility of supporting very large transaction rates (millions of reads and writes per second) with good response time (less than a millisecond) using the DSH-11 storage hierarchy and the read-through and store-behind algorithms.

A GPSS/360 simulation model is developed for a DSH-11 configuration with one processor and three storage levels. The results obtained from this model are very interesting. It is found that, at very high locality levels, when most of the references are satisfied by the highest performance storage level, the store-behind algorithm interacts with the DSH-11 buffer management algorithms to create a system deadlock. This has not been anticipated in the design of DSH-11, and has led to a redesign of the DSH-11 buffer management scheme.

Another GPSS/360 simulation model is developed for a DSH-11 configuration with five processors and four storage levels. This model makes use of deadlock-free buffer management algorithms. Results from this model reveal further interesting properties of the store-behind algorithm and of the DSH-11 design. It is found that at high locality levels, the store-behind requests form a pipeline. Thus the rate of write operations that can be serviced is limited by the slowest stage in the pipeline, i.e., the slowest storage device. It is also found that a bottleneck may be developed at the lowest level when the block size of that level is too large.

A well-balanced system is obtained by increasing the degree of parallelism in the lower storage levels and by decreasing the block sizes used by these storage levels. This system is then used as a basis to compare the performance of the DSH-11 architecture under different technology assumptions. It is found that using 1979 technologies, a throughput of .7 million operations per second with mean response time of 60 microseconds are obtained for a mix of storage references consisting of 30 percent read requests. Using 1985 technologies, the same storage reference mix produces a throughput of 4 million operations per second with 10 microseconds mean response time.

6.2 A SIMULATION MODEL OF DSH-11 : THE P1L3 MODEL

The P1L3 model of DSH-11 is a GPSS/360 model of a DSH-11 configuration with one processor and three storage levels. It represents a basic structure from which extensions to include more processors and storage levels can be made. The structure of P1L3 is illustrated in Figure 6.1(a). Each module in Figure 6.1(a) actually consists of four queues and a facility (Figure 6.1(b)). The facility is referred to as the request processor (RP). There are two input queues, one for transactions with data (the XQ), and one for transactions with messages (the IQ). The two corresponding output queues are named YQ and OQ respectively. The XQs and YQs have limited capacity, since they are the data buffers. There is no limit on the lengths of the IQs and the OQs. The following example illustrates the naming conventions used in the model. The K2 module actually consists of the KRP2, KIQ2, KOQ2, KXQ2 and KYQ2. The current length of KXQ2 is denoted as KXL2 and the maximum allowable length of KXQ2 is denoted as KXM2.

6.2.1 An Illustration of the DSH-11 Algorithms

A listing of the P1L3 model is presented in Appendix A. To illustrate the model logic, the following is a brief description of the path followed by a read-through transaction. A read request (TXN) is queued in KIQ3 (the input message

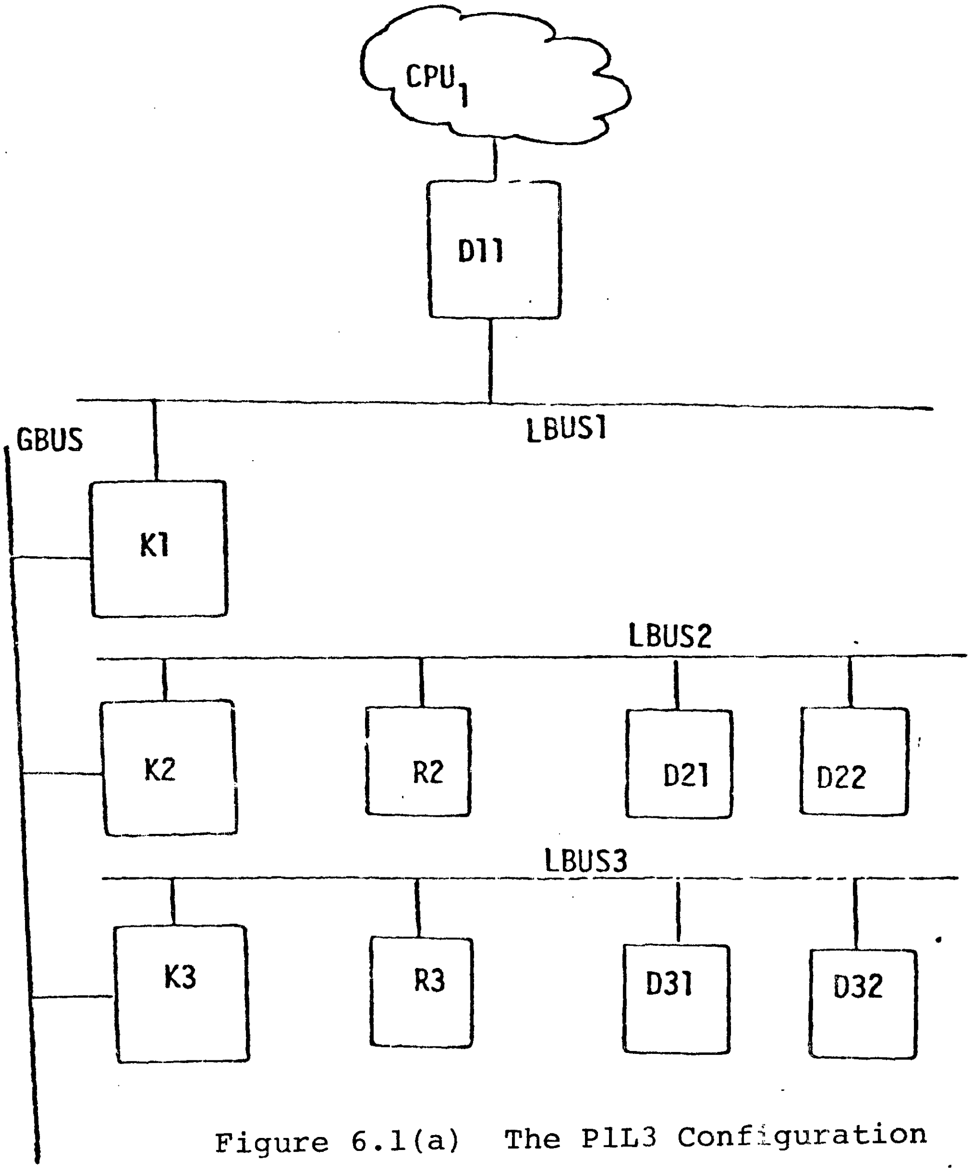


Figure 6.1(a) The P1L3 Configuration

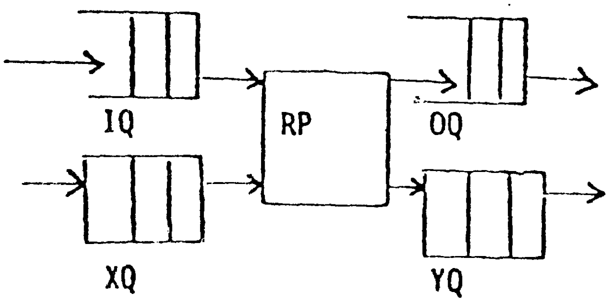


Figure 6.1(b) A Module in P1L3

DEGREE OF MULTIPROGRAMING OF A CPU = 20

SIZES OF DATA QUEUES (XQ AND YQ) = 10

DIRECTORY SEARCH TIME = 200 NANOSEC.

READ/WRITE TIME OF A L(1) STORAGE DEVICE = 100 NANOSEC.

READ/WRITE TIME OF A L(2) DEVICE = 1000 NANOSEC.

READ/WRITE TIME OF A L(3) DEVICE = 10000 NANOSEC.

BUS SPEED = 10 MHZ

BUS WIDTH = 8 BYTES

SIZE OF A TRANSACTION WITHOUT DATA = 8 BYTES

BLOCK SIZE AT L(1) = 8 BYTES

BLOCK SIZE AT L(2) = 128 BYTES

BLOCK SIZE AT L(3) = 1024 BYTES

% READ REQUESTS = 70%

% WRITE REQUESTS = 30%

CONDITIONAL PROB. OF FINDING DATA IN A LEVEL
GIVEN THAT THE DATA IS NOT IN ANY UPPER LEVEL = P

Figure 6.2 Input Parameters to P1L3

queue of the storage level controller at level 3). When KRP3 is free, TXN is serviced and put into KOQ3. When LBUS3 is available, TXN is sent to RIQ3 (the input message queue of the memory request processor at level 3) where it waits for RRP3, the request processor. RRP3 then searches its directory to obtain the real address for TXN. TXN is put into ROQ3 to be sent to a storage device, say, D31. When LBUS3 is free, TXN is sent to DIQ31 (the input message queue for device D31). TXN waits in DIQ31 for DRP31 to be free and also for a slot in DYQ31 (the output data queue for D31) to hold the retrieved data. When both conditions are met, DRP31 retrieves the data and puts it in DYQ31 where it waits for the LBUS3 to be free and for there to be a slot in KXQ3 (the input data queue of the storage level controller at level 3) to hold the data. When both conditions are met, the data is sent to KXQ3. Then the data is put in KYQ3 waiting for the GBUS and for all the upper storage levels to be free to receive the broadcast...

6.2.2 The P1L3 Model Parameters

The model is highly parametized. Parameters for the P1L3 model are chosen to reflect current (1979) processor and storage technology. A key parameter that characterizes the references made to DSH-11 is the locality level. The locality level (P) is the condition probability that a reference is satisfied at a given storage level given that the reference is not satisfied in all upper storage levels. Figure 6.2 summarizes all the model parameters.

6.3 SIMULATION RESULTS OF THE P1L3 MODEL

Three different locality levels are used for the P1L3 model. The simulated time is one milisecond (one million time units in the model). Some unusual phenomena are uncovered during the analysis of these preliminary results. This leads to more extensive simulation studies to obtain more data points. A plausible theory is then proposed to explain these phenomena. Detail traces of the model is used to verify the theory. The findings are discussed in the following subsections.

6.3.1 Preliminary Studies Using the P1L3 Model

A series of three simulation studies are carried out with three locality levels : high (P=.85), medium (P=.5), and low (P=.2). Throughputs, mean response times and utilizations of the facilities are summarized in Figure 6.3.

Throughput in millions transactions per second are plotted against the locality levels in Figure 6.4. From Figure 6.4, it seems that a throughput of .6 million transactions per second is the maximum that one could obtain with this configuration.

Locality Level	Throughput per millisecond	GBUS Utilization	LBUS1 Utilization	Data Cache Utilization	LBUS2 Utilization	D21 Utilization	LBUS3 Utilization	D31 Utilization	Mean Response Time (Nsec)
.85	598	.23	.05	.19	.26	.09	.35	.52	6021
.50	548	.50	.09	.19	.65	.17	.62	.99	31324
.20	286	.41	.07	.10	.63	.11	.52	1.00	64032

Figure 6.3 Preliminary Results of the P1L3 Model

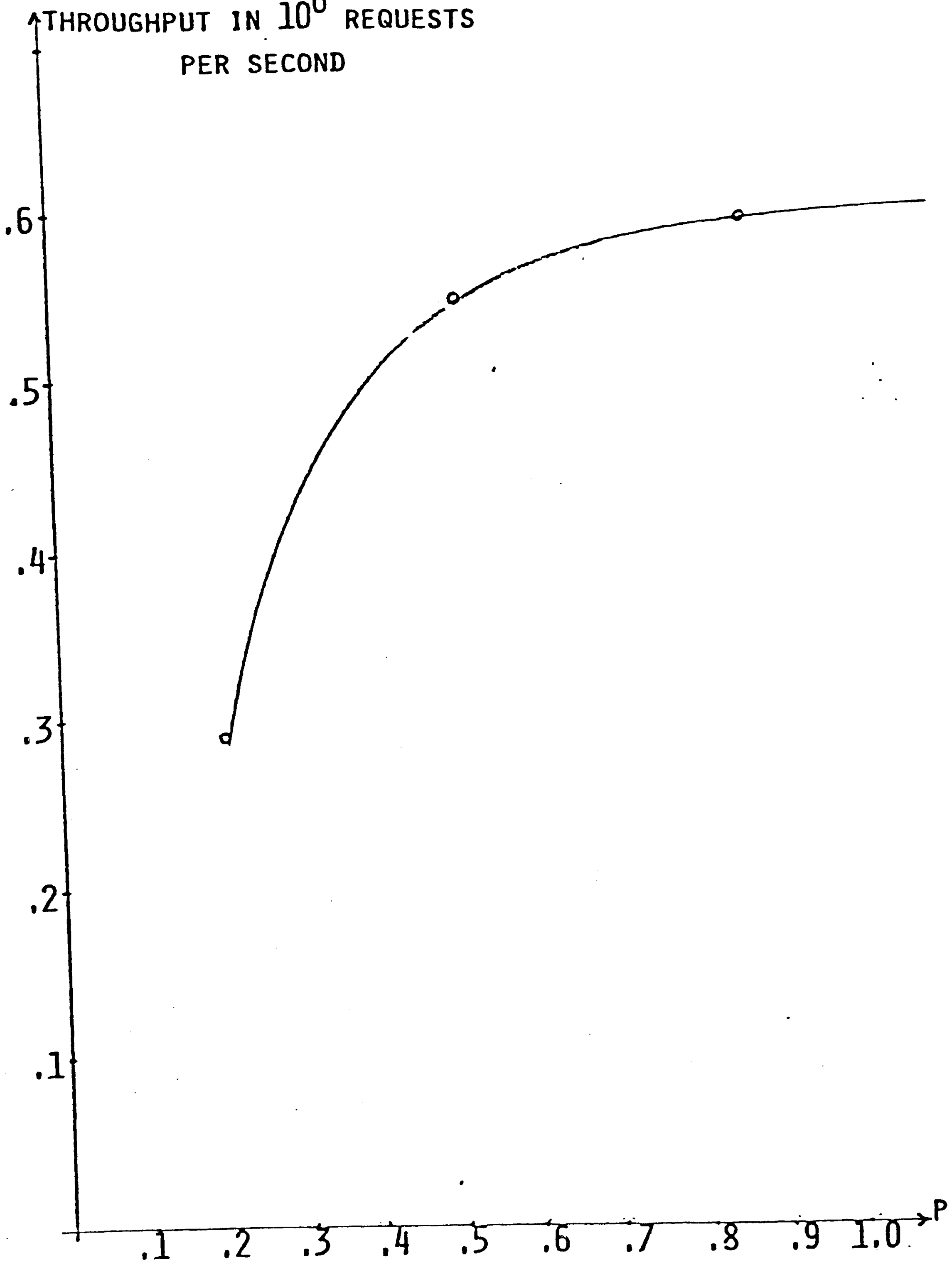


Figure 6.4 Throughput Vs. Locality Level
(P1L3 Preliminary Results)

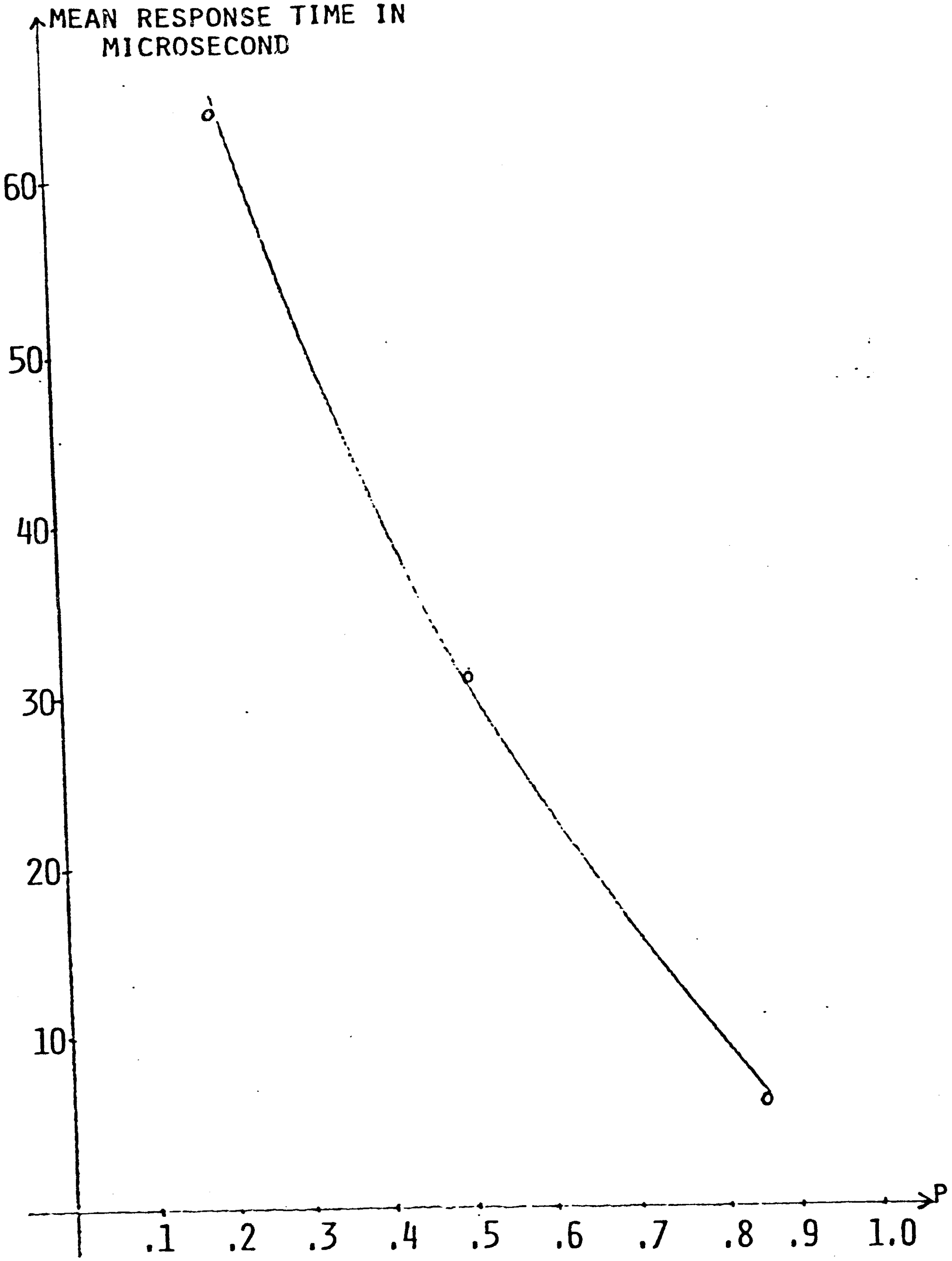


Figure 6.5 Mean Response Time Vs. Locality Level
(P1L3 Preliminary Results)

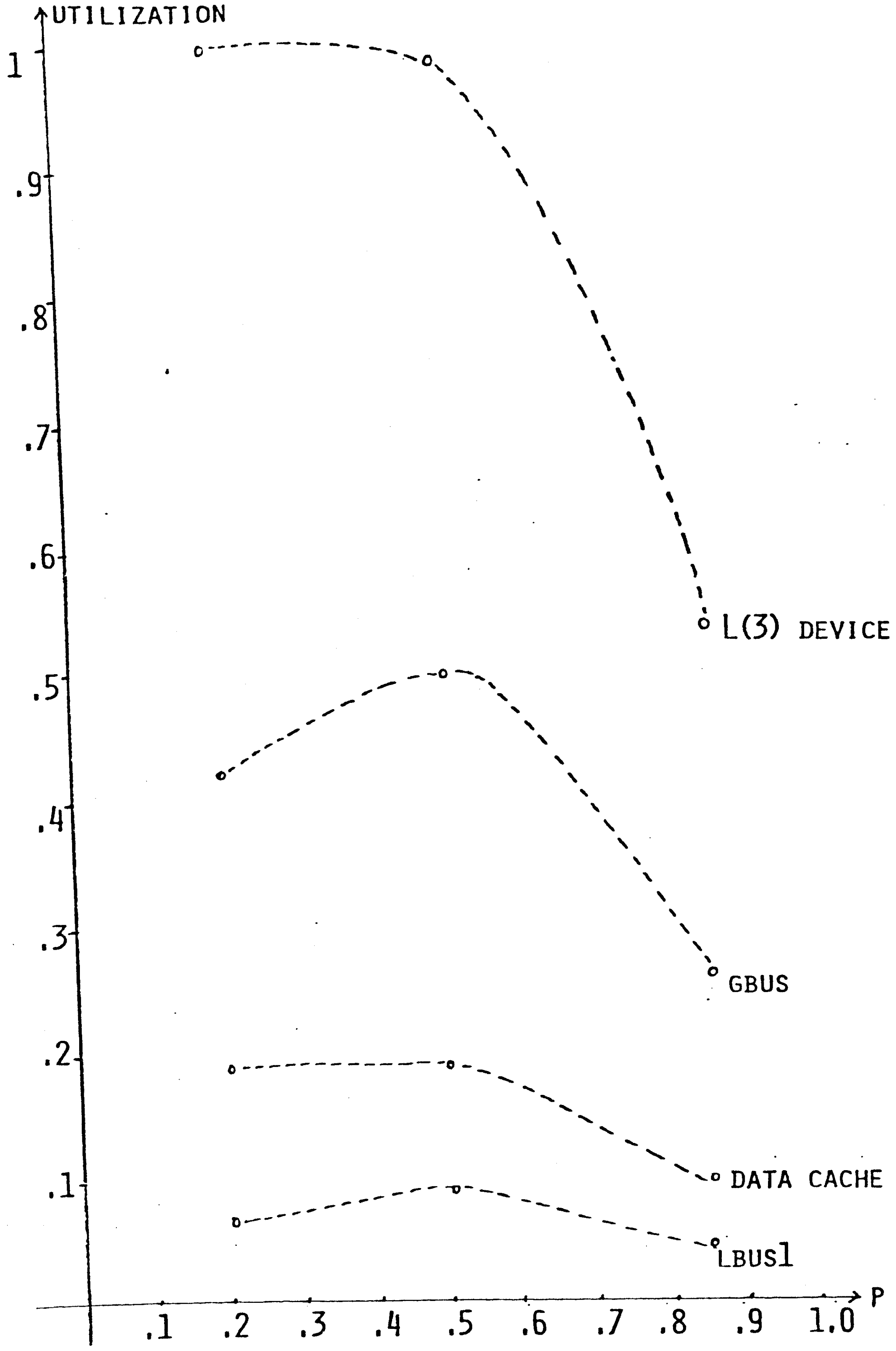


Figure 6.6 Utilization Vs. Locality Level
(P1L3 Preliminary Results)

Mean response time per transaction are plotted against locality levels in Figure 6.5. This shows that a mean response time of 5 micro seconds is obtainable at high locality levels. Furthermore, as the locality level increases, there will be more references being satisfied in the high performance storage levels, thus the mean response time will decrease.

Utilizations of the various facilities are plotted against locality levels in Figure 6.6. It can be seen from these plots that at low locality levels, the slowest storage level becomes a system bottleneck. At higher locality levels, bus utilizations drop because most references are satisfied by the data cache, DRP11, making the use of the buses unnecessary except for store-behind operations.

At high locality levels, one would also expect the utilization of the data cache, DRP11, to be very high. However, this is not supported by the data. In fact, even though the throughput at the P=.85 locality level is larger than that at the P=.5Ø locality level, the DRP11 utilization actually drops.

Examine the data more closely, another puzzle is discovered. If one multiply throughput by the mean response time divided by the maximum degree of multiprogramming, one should obtain a number closed to the simulated time (1,000,000). For the P=.20 case, this number comes out to be 915657. For the P=.50 case, this number comes out to be 858277. But for the P=.85 case, this number is only 180027. It is suspected that either the data is wrong or there are some unusual blocking phenomena in the system in the P=.85 case.

6.3.2 More Extensive Studies Using the P1L3 Model

Since it is not difficult to obtain more data points by varying the locality levels, a second series of simulations is carried out. The results of these simulations are presented in Figure 6.7.

Throughputs are plotted against locality levels in Figure 6.8. This shows that as the locality level increases, throughput also increases. A throughput of closed to one million transactions per second is obtainable at about P=.80 locality level. However, after the P=.80 point, throughput drops sharply as the locality level increases. This requires some explaination.

In Figure 6.9, mean response time is plotted against locality levels. This shows that as locality level increases, mean response time decreases. This plot does not seem to provide insight as to why throughput decrease sharply after the P=.80 locality level.

Locality Level	Throughput/ millisecond	GBUS Utilization	LBUS1 Utilization	Data Cache Utilization	LBUS2 Utilization	D21 Utilization	LBUS3 Utilization	D31 Utilization	Mean Response Time in nanosecond
.95	532	.14	.03	.15	.16	.05	.21	.25	3986
.90	581	.16	.04	.17	.19	.06	.26	.42	3957
.85	589	.23	.52	.19	.26	.09	.35	.52	6021
.80	947	.50	.94	.31	.57	.19	.71	.99	16298
.70	811	.53	.10	.27	.65	.20	.69	1.00	23317
.65	758	.51	.10	.26	.62	.18	.68	1.00	22505
.60	698	.51	.10	.23	.63	.19	.65	1.00	27114
.50	548	.50	.10	.20	.65	.17	.62	1.00	31324
.40	456	.45	.08	.16	.63	.15	.59	1.00	39142
.30	320	.42	.07	.12	.60	.12	.52	1.00	56908
.20	286	.42	.07	.10	.63	.11	.52	1.00	64032

Figure 6.7 P1L3 Model - Extensive Results

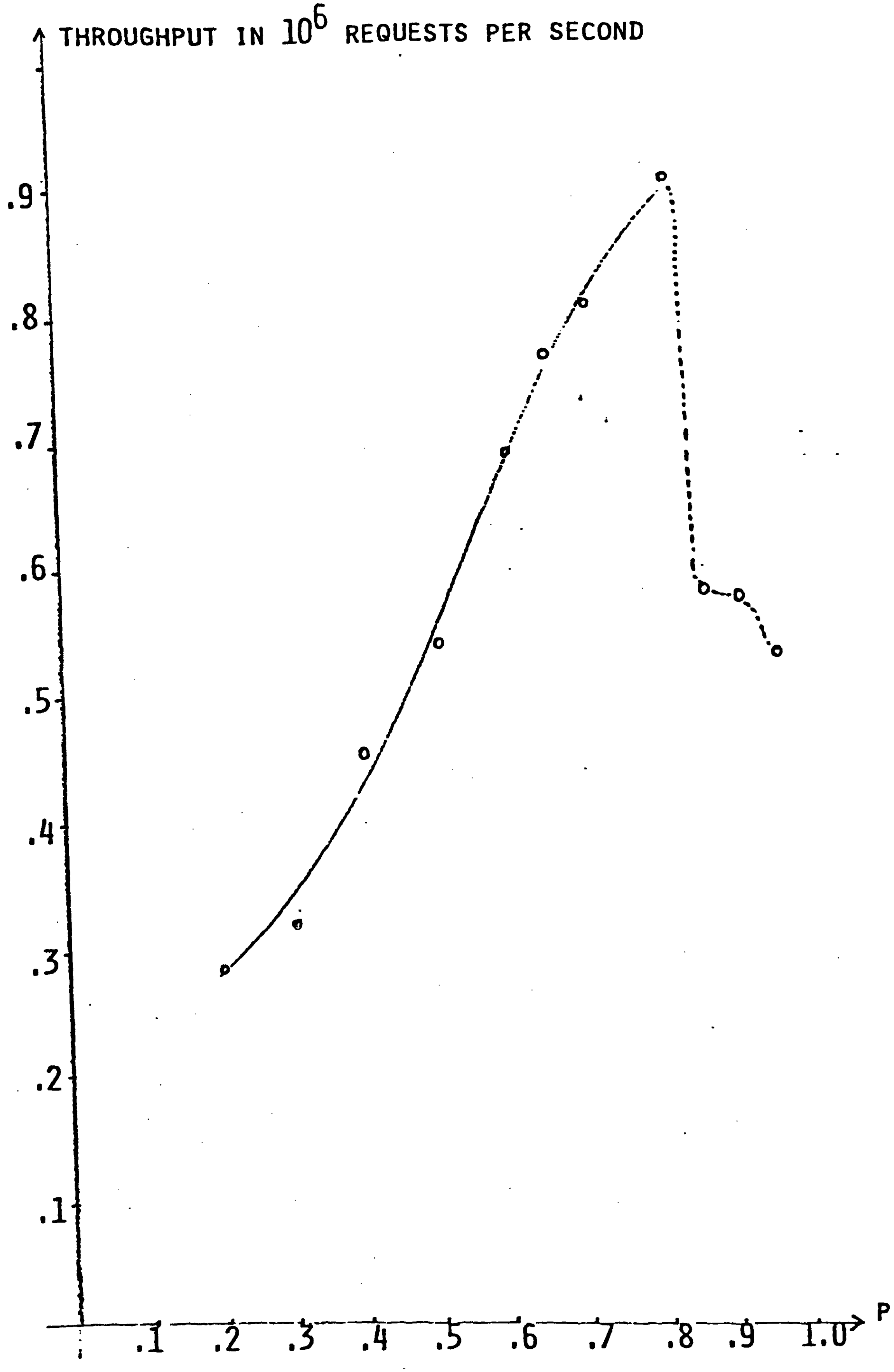


Figure 6.8 Throughput Vs. Locality Level
(P1L3 Extensive Results)

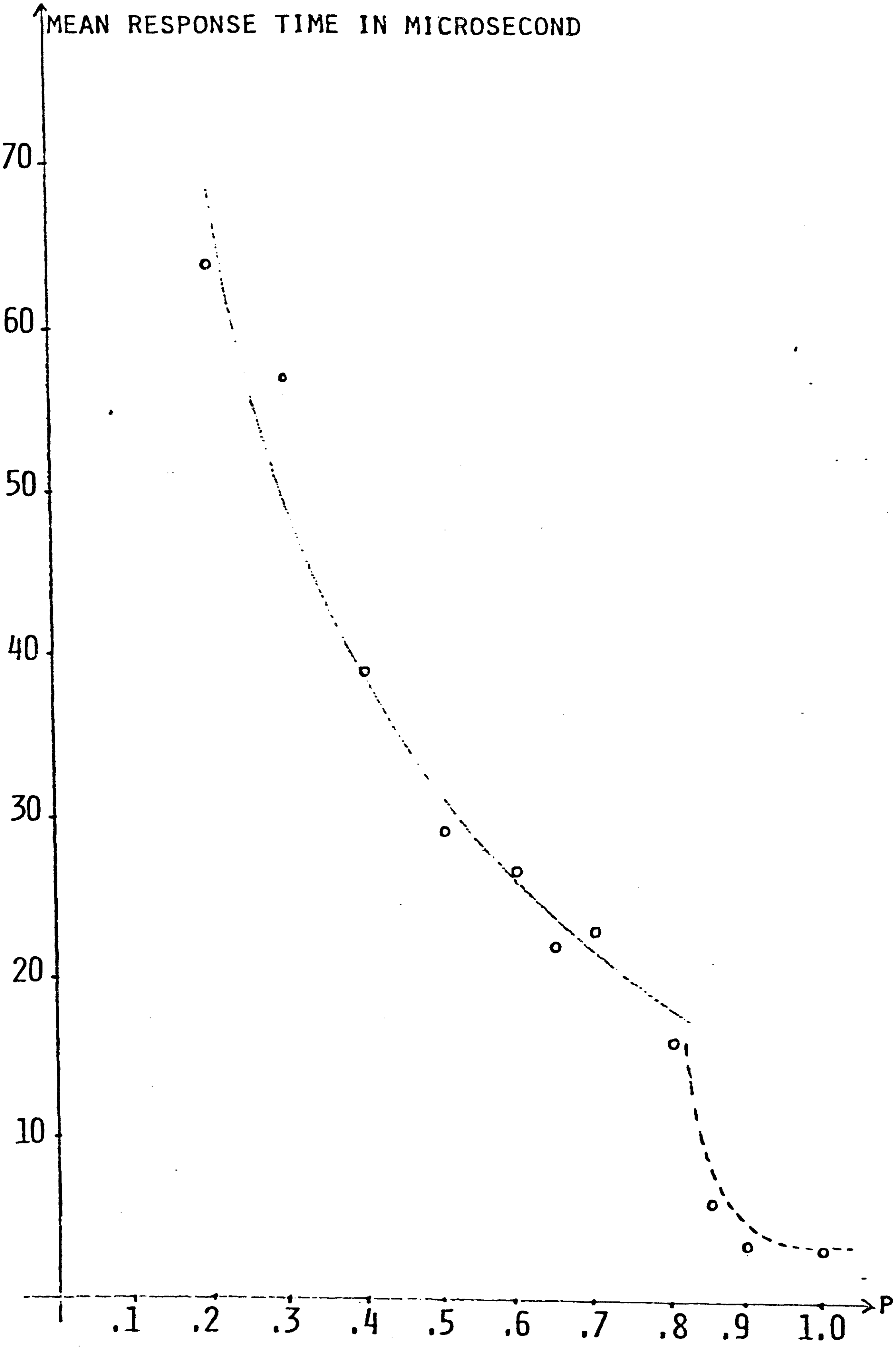


Figure 6.9 Mean Response Time Vs. Locality Level (P1L3 Extensive Results)

6.3.3 A Plausible Theory of Operation

One theory to explain the sharp drop in throughput at very high locality levels is that at such high locality levels, the rate of write operations being generated is very high. Since a write will not proceed until DRP11 is free (to write the data), and DRP11's YQ has a buffer slot (for holding the store-behind operation), the write operation may hold up other transactions in their use of DRP11. Since the utilization of DRP11 is very low, the blocking must be due to the YQ being full often. Many store-behind transactions closed togther will tend to make the YQ full often. These blocking transactions will tend to hold up other transactions hence resulting in low system throughput.

If the YQ is full often, it must be because transactions in it cannot move on to the next facility. This will happen if the bus LBUS1 is busy or the XQ buffer of K1 is full, or both. From the data, we see that all the bus utilizations are very low, hence the blocking must be due to the fact that the XQ buffer of K1 is full often. Proceeding in this manner, one could argue that at high locality levels, the rate of store-behind operations is very high, which results in store-behind transactions being backed up from a storage device. This backing up of store-behind operations causes long queueing delays for other transactions as well, resulting in low system throughput. This blocking situation also prevents DRPll to be kept busy as evident from its low utilization.

We can now explain why the utilization of DRPll at the P=.85 locality level is lower than that at the P=.50 locality level. At P=.85, due to the store-behind transactions being backed up, very few acknowledgements to the store-behind transactions ever return to DRPll. In the P=.50 case, most acknowledgements to store-behind transactions return to DRPll. Thus, even though the number of reads and writes handled by DRPll in the P=.50 case is lower than that handled by the DRPll in the P=.85 case, there are many more acknowledgements serviced by DRPll in the P=.50 case, hence the corresponding utilization is higher.

There are no backing up of store-behind transactions in the low locality level cases because the rate at which they are generated is low. Since the store-behind transactions are separated from one another there is enough time for a device to service a previous store-behind transaction before another one comes along.

6.3.4 Verification of Theory with Data

The above theory seems to explain the phenomena well and agrees well with the observed data. To verify the theory, detail model traces are examined to determine the status of the system at the time of simulation termination.

It is found that for low locality levels, there is indeed no backing up of the store-behind transactions. There is a backlog of requests to be processed by the lowest storage level devices due to their large service times. For high locality levels, starting from P=.85, store-behind transactions begin to be backed up, from storage level 2. However, the back up is due to a system deadlock developed at storage level 2, and not due to the slower speeds of the devices, as hypothesized above.

The deadlock at storage level 2 is illustrated in Figure 6.10. All the XQs and YQs are full. A store-behind transaction in DYQ21 is waiting for LBUS2 and a KXQ2 buffer slot. LBUS2 is free but KXQ2 buffer is full. KXQ2 will not be cleared because KYQ2 is full. KYQ2 cannot be cleared because both buffer of R2 are full. These buffers cannot be cleared because DXQ21 and DYQ21 are full. DYQ21 cannot be cleared because it is waiting for KXQ2 to be cleared. Thus a deadlock is developed. This deadlock causes the XQs and YQs in the upper storage levels to be gradually filled as

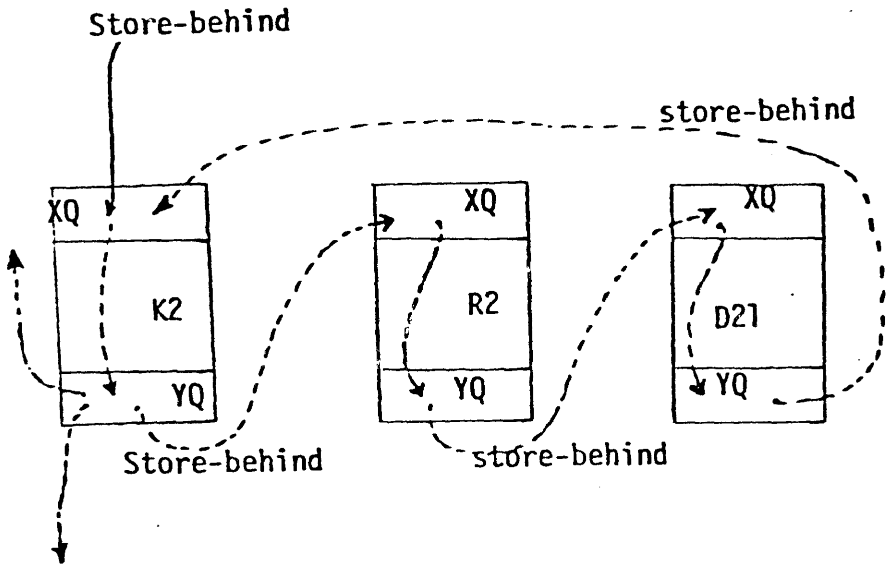


Figure 6.10 Deadlock In P1L3 Model

more store-behind transactions are generated. When YQ at DRP11 is full, the system will be held up when the next write transaction arrives.

It is interesting to note that this deadlock only occurs at very high locality levels. This is beacuse at high locality levels, the rate of store-behind transactions generated is very high. Comparing the P=.95 case and the P=.50 case, even though the same number of store-behind transactions are generated to lower storage levels in both cases, the rate at which they are generated in the P=.95 case is 30 times that of the P=.50 case. Store-behind transactions sparsely separated from one another give chance for the device to service them, therefore avoiding a deadlock. This deadlock situation is not too different from a traffic jam at a Boston rotary during rush hour.

In retrospect, the causes of the deadlock are due to the rate of store-behind transactions and the use of one single buffer for data coming into a storage level as well as for data going out of a storage level. The potential for deadlock of using a common buffer was not discovered during the design of DSH-11 due to the complex interactions of the various protocols for store-behind, read-through, and overflow handling operations.

6.4 DEADLOCK-FREE BUFFER MANAGEMENT SCHEMES

In the DSH-11 simulation model, there are five types of transactions supporting the read-through and store-behind operations. These transactions are : read-through-request (RR), read-through-result (RT), overflow (OV), store-behind-request (SB), and acknowledgement (AK). Each type of transaction is handled differently. Furthermore, the same type of transaction is handled differently depending on whether the transaction is going into or out of a storage level. A potential deadlock exists when different transactions share the same buffer and their paths form a closed loop. We have seen an example of such deadlock in the P1L3 model where SB transactions coming into a storage level and SB transactions going out of a storage level form a closed loop. Other potential deadlocks exists in the P1L3 model. This section is focused on developing deadlock-free buffer management algorithms.

Potential deadlocks exist beacause different transaction types share the same buffer and that the First Come First Serve (FCFS) strategy is used for allocating buffer slots. A simple strategy to avoid deadlock is not to allow buffer sharing among different transaction types. No path crossing can occur thus no loop can exist. Although this strategy is easy to implement, it does not make optimal use of the buffer space. Another strategy to avoid deadlock is to allow buffer sharing, but to make use of more sophisticated buffer allocation algorithms. One such algorithm is discussed below.

6.4.1 A Deadlock-free Buffer Allocation Algorithm

Two types of buffers are used at each storage level, the IN buffers and the OUT buffers. Transactions coming into the storage level use the IN buffers and transactions going out of the storage level use the OUT buffers. Transaction coming into a storage level from a higher storage level are the RR, SB, and OV transactions. Transactions coming into a storage level from a lower storage level are the RT and AK transactions. Similarly, transactions going out of a storage level to the next lower storage level are the RR, SB, and OV transactions. Transactions going out of a storage level to a higher storage level are the RT and AK transactions. Each component in a storage level has an IN buffer and an OUT buffer. This is illustrated in Figure 6.11.

The general idea of this buffer allocation scheme is not to allow the buffers to be completely filled. When the buffers are filled up to a certain level, only those transactions that can be processed to completion and resulting in freeing up buffer slots are accepted. The precise algorithm is as follows.

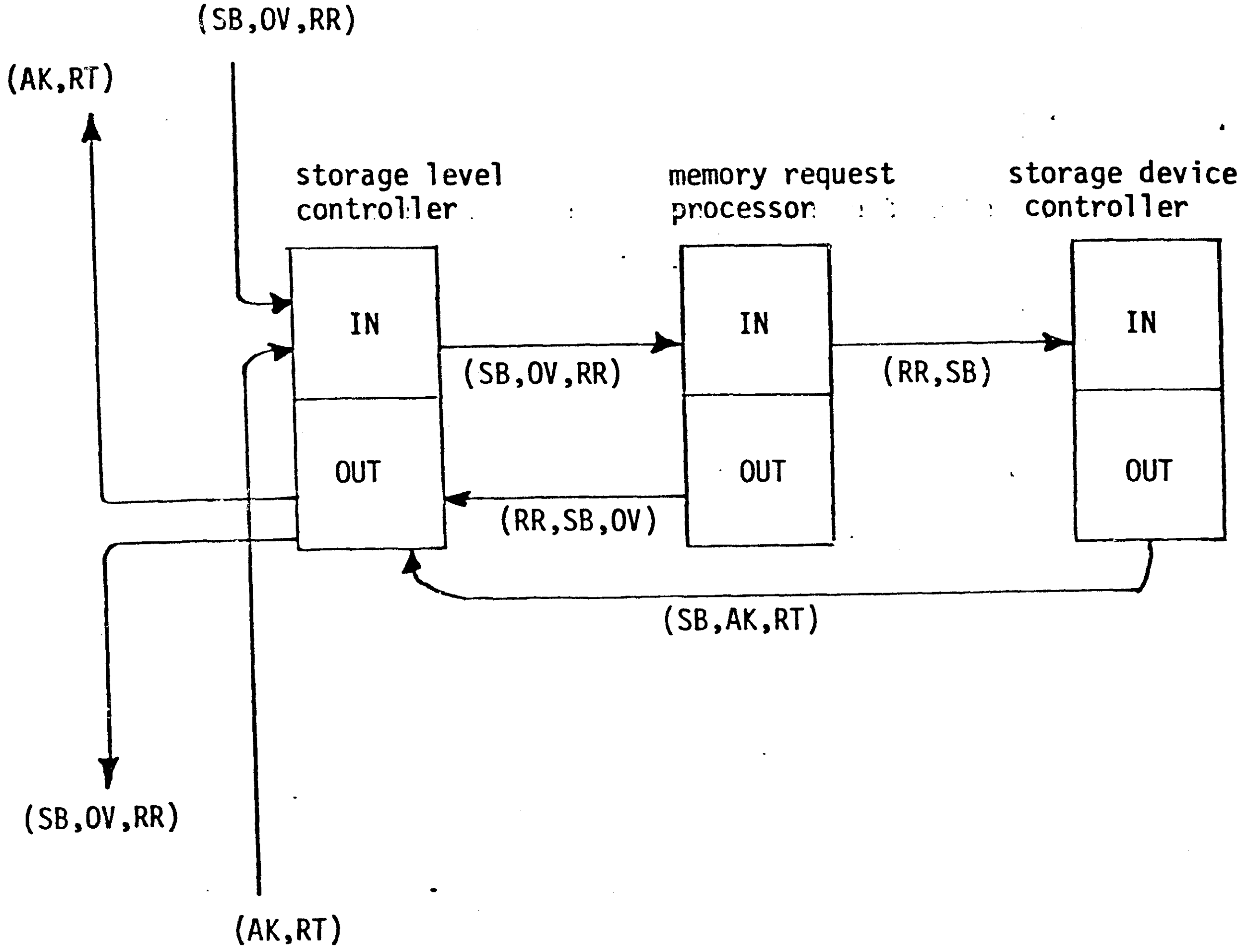


Figure 6.11 A Deadlock-free Buffer Scheme

1. The size of OUT is always greater than the size of IN.

2. Always maintain at least one empty slot in an IN buffer.

3. Buffer-full (BF) condition is raised when the number of transactions in IN plus the number of transactions in OUT is equal to the size of OUT.

4. If BF then do not accept any RR or SB into a storage level. Only process OV, RT, and AK transactions.

We now provide an informal argument to show that the scheme described above is indeed deadlock-free. First we have to show that the RR and SB transactions are not the only transactions in the system when all the buffer pairs have their BF conditions raised. Then we have to show that processing each of the OV, AK and RT transactions will free up some buffer slots thus lowering some BF conditions.

Suppose that all the BF conditions are raised. Examine the OUT buffers of the lowest storage level. Since the size of OUT is greater than that of IN, BF implies that there is at least one transaction in OUT. This transaction must be going out of the storage level to a higher storage level, hence cannot be a RR or a SB transaction.

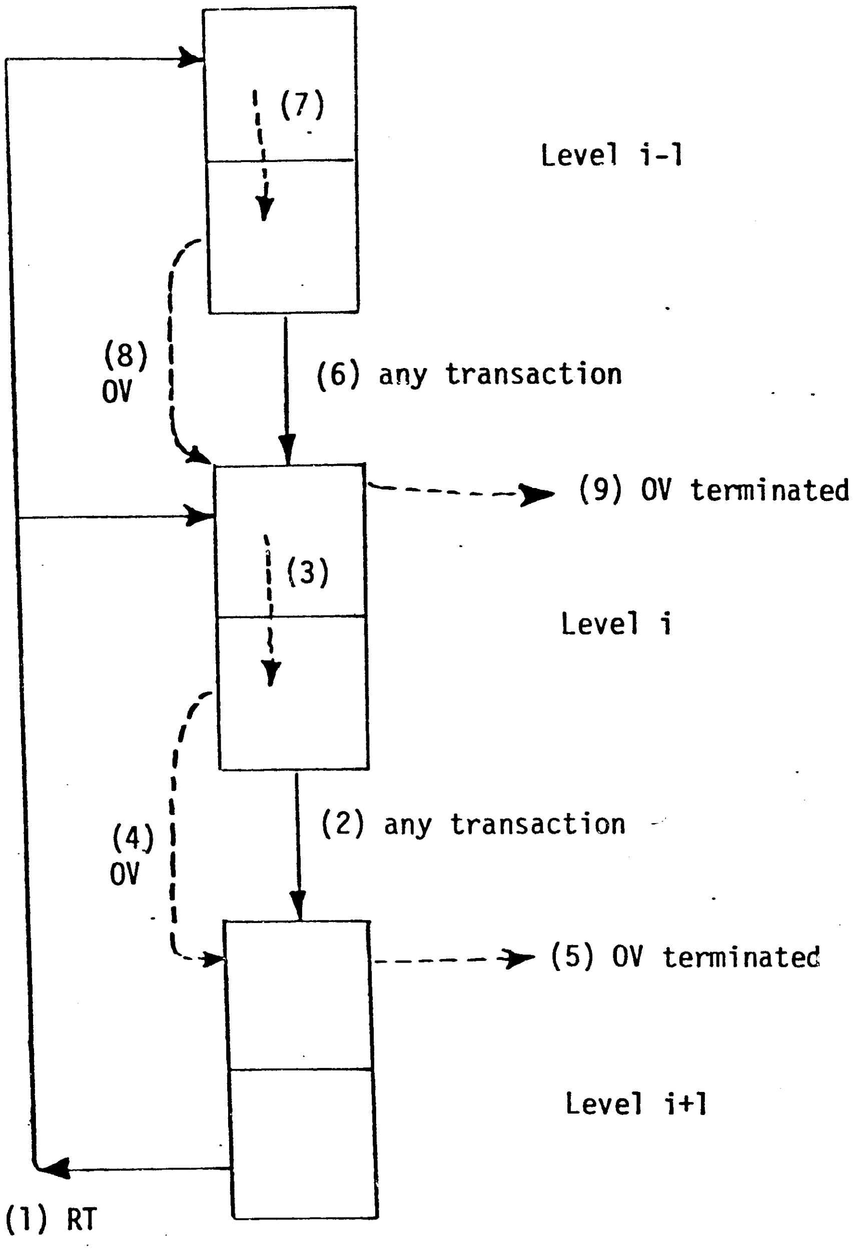


Figure 6.12 Illustration of the Deadlock-free Buffer Scheme

Consider a RT transaction at level i+1 (Figure 6.12). (1) All upper storage level, level i and level i-1 can receive this transaction since there is always one empty slot in each IN buffer. The departure of the RT transaction creates an empty slot in the OUT buffer of the sender (level i+1). (2) Level i can now send a transaction to level i+1 which creates a slot in level i. The RT transaction can now be serviced in level i. (3) Handling the RT transaction may create an OV transaction in level i. Luckily there is a buffer slot for the OV transaction in level i. (4) The OV transaction can be sent to level i+1 because there is always a free slot in the IN buffer at level i+1. (5) The OV transaction will be serviced to completion in level i+1. Hence, there is a free slot in level i as result of these operations. (6) Now a transaction from level i-1 can be sent to level i. (7) The RT transaction can be handled in level i-1 which may create an OV transaction. (8) The OV transaction can be sent to level i. (9) Finally, the OV transaction is handled and terminated in level i. Thus, there is a free buffer slot created in level i-1 as a result of processing the RT transaction.

Handling an AK transaction may generate another AK to be sent to the immediate upper storage level. The same argument for the RT transaction can be applied to show that a buffer slot will be freed up as a result of handling the AK transaction.

It is clear from the above discussion that this buffer management scheme requires more complex protocols among storage levels and a complex priority scheme for the transactions. A key advantage of this scheme is that it makes efficient use of buffer space since different transactions with varying buffer space requirements can share a common buffer pool.

6.5 ANOTHER SIMULATION MODEL OF DSH-11 : THE P5L4 MODEL

A GPSS/360 simulation model of another DSH-11 configuration with five processors and four storage levels is developed. This model is referred to as the P5L4 model. This model revised the basic logic used in the P1L3 model to use a deadlock-free buffer management scheme and to accomodate four additional processors and an additional storage level. The simple scheme of using separate buffers for different transactions is used for the P5L4 model.

The first series of studies provides further insights to the operation of the store-behind algorithms. It also shows that level 4 storage may be too slow and its local bus may not have engough bandwidth to support the amount of data transfer activities at that level.

The second series of studies is aimed at obtaining a well-balanced system. The degree of parallelism in the lower storage levels are increased and the demand on the buses is lowered by reducing the block sizes. A well-balanced system is obtained which is then used as the basic system to study the effect of using projected 1985 technologies for DSH-11. Results of these studies and their analysis are presented in the following sections, after a brief introduction to the P5L4 model and its parameters.

6.5.1 The P5L4 Model and its Parameters

The structure of the P5L4 model is very similar to that of the P1L3 model. However, the basic component of the model is quite different. The basic component of the P5L4 model is a facility and a number of data buffers, one for each type of transaction comming into the storage level and going out of the storage level. Figure 6.13(a) illustrates the DSH-11 configuration that P5L4 is modelling, and Figure 6.13(b) illustrates the basic component of the model. A flow chart of the P5L4 model logic is presented in Appendic B. A listing of the P5L4 model is presented in Appendix C.

The parameters used in the P5L4 model are the same as those used in the P1L3 model with the following exceptions. (1) There are five processors, each with 1Ø degrees of mul-

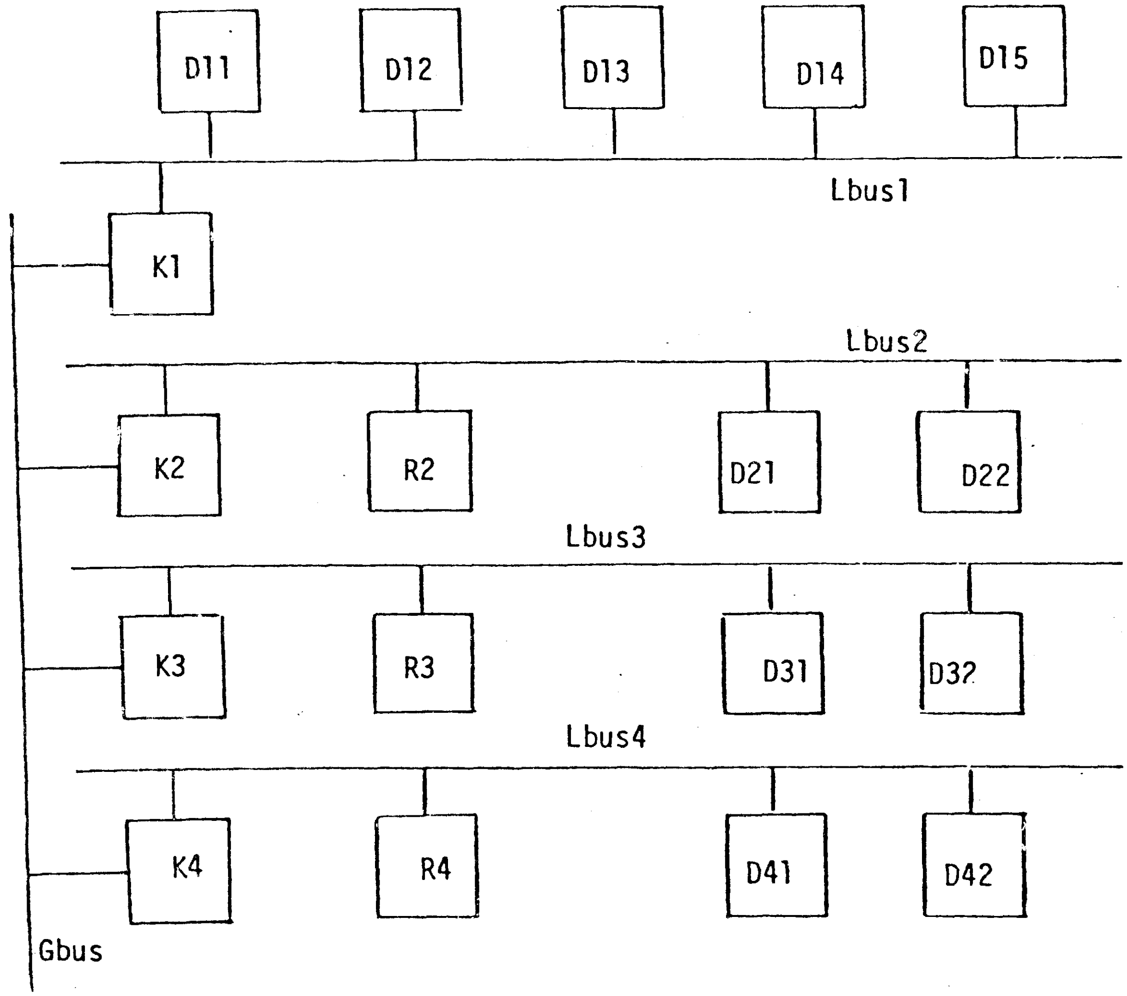


Figure 6.13(a) The P5L4 Configuration

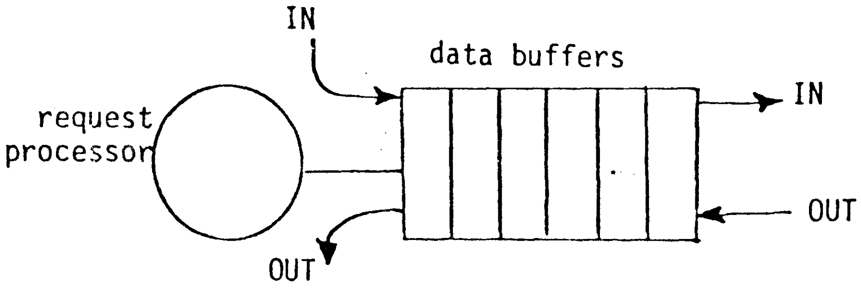


Figure 6.13(b) A Module in P5L4

DEGREE OF MULTIPROGRAMING OF A CPU = 10

SIZES OF DATA BUFFERS = 10

DIRECTORY SEARCH TIME = 200 NANOSEC.

READ/WRITE TIME OF A L(1) STORAGE DEVICE = 100 NANOSEC.

READ/WRITE TIME OF A L(2) DEVICE = 1000 NANOSEC.

READ/WRITE TIME OF A L(3) DEVICE = 10000 NANOSEC.

BUS SPEED = 10 MHZ

BUS WIDTH = 8 BYTES

SIZE OF A TRANSACTION WITHOUT DATA = 8 BYTES

BLOCK SIZE AT L(1) = 8 BYTES

BLOCK SIZE AT L(2) = 128 BYTES

BLOCK SIZE AT L(3) = 1024 BYTES

% READ REQUESTS = 70%

% WRITE REQUESTS = 30%

CONDITIONAL PRCB. OF FINDING DATA IN A LEVEL
GIVEN THAT THE DATA IS NOT IN ANY UPPER LEVEL = P

Figure 6.14 Input Parameters to the P5L4 Model

tiprogramming (as opposed to 20 in the P1L3 model). (2) There is a new storage level with 2 storage devices having access times 10 times higher than those of the devices in level 3. The parameters used in the P5L4 model are summarized in Figure 6.14.

6.5.2 Preliminary Studies Using the P5L4 Model

A preliminary study using the P5L4 model is carried out using several different locality levels and using the parameters listed in Figure 6.14. The simulated time is one millisecond (one million model time units). Results from these studies are summarized in Figure 6.15. Figure 6.15(a) is a table listing the throughput, mean response time, total transaction wait time, total transaction execution time, and 'system utilization'. System utilization is defined as the ratio of the product of the total number of transactions completed and the mean response time to the product of the simulated time and the maximum number of active requests pending at all the processors. It indicates the percentage time that DSH-11 is busy.

Figure 6.15(b) tabulates the utilizations of the buses and the utilizations of typical storage devices at each storage level. The utilizations of all the memory request processors and all the the storage level controllers are very

locality level	throughput (per ms.)	mean response time (ns)	total wait time (ms)	total exec time (ms)	system utilization
.50	418	45805	17450	1690	.38
.60	600	35442	19910	1360	.43
.70	717	46664	32490	970	.67
.80	782	43570	32230	840	.68
.95	788	40148	31360	270	.63

(a) Throughput, Response Time, etc.

(b) Utilizations

P	gbus	lbus1	lbus2	lbus3	lbus4	L1	L2	L3	L4
.50	.92	.06	.30	.99	.94	.02	.11	.46	.99
.60	.91	.06	.31	.99	.94	.03	.11	.46	.99
.70	.91	.07	.32	.99	.94	.04	.13	.52	.88
.80	.88	.05	.26	.99	.94	.04	.10	.44	.92
.95	.84	.04	.21	.99	.95	.04	.06	.51	.90

Figure 6.15 Preliminary Results of P5L4 Model

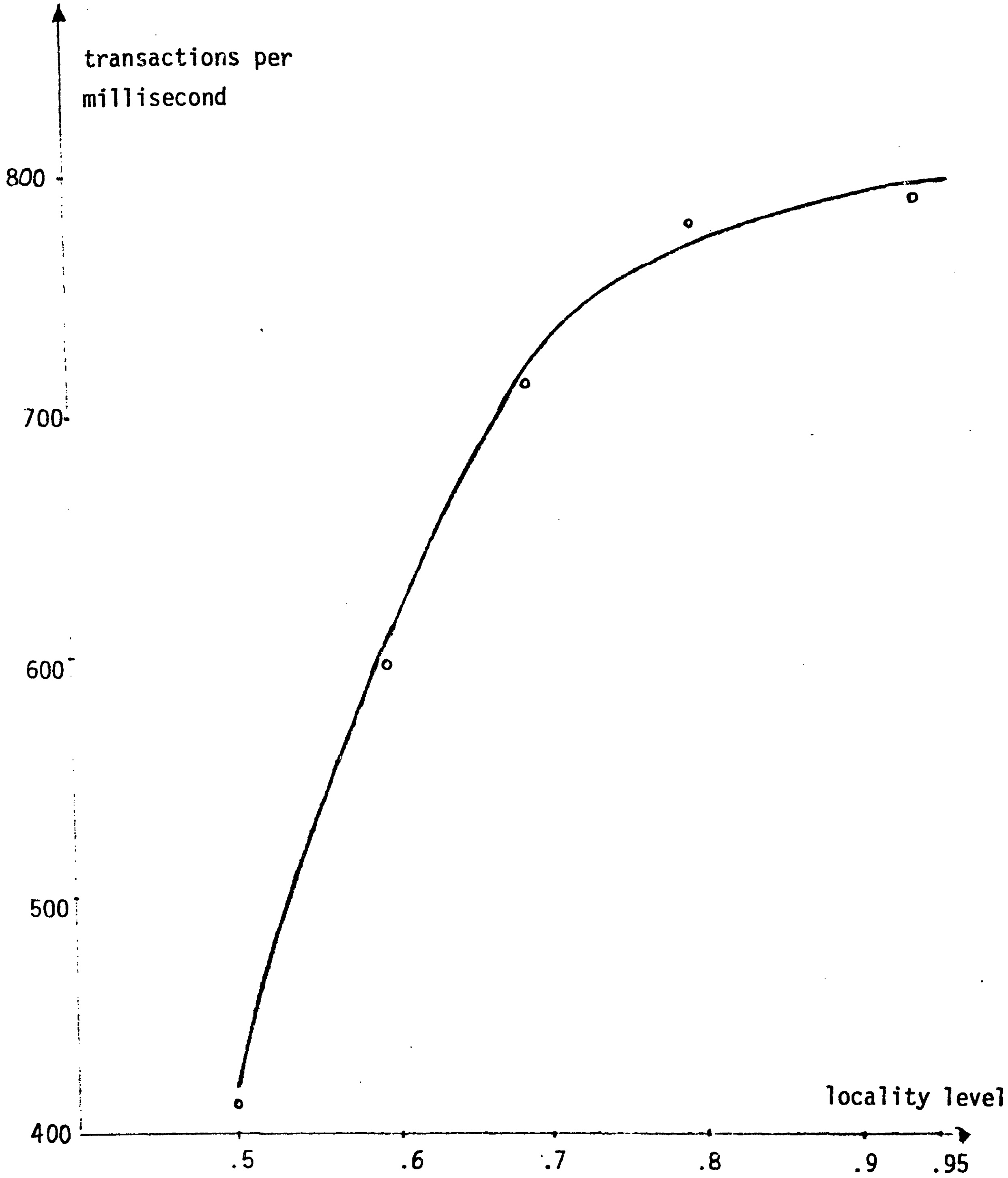


Figure 6.16 Throughput Vs. Locality Level
(P5L4 Preliminary Results)

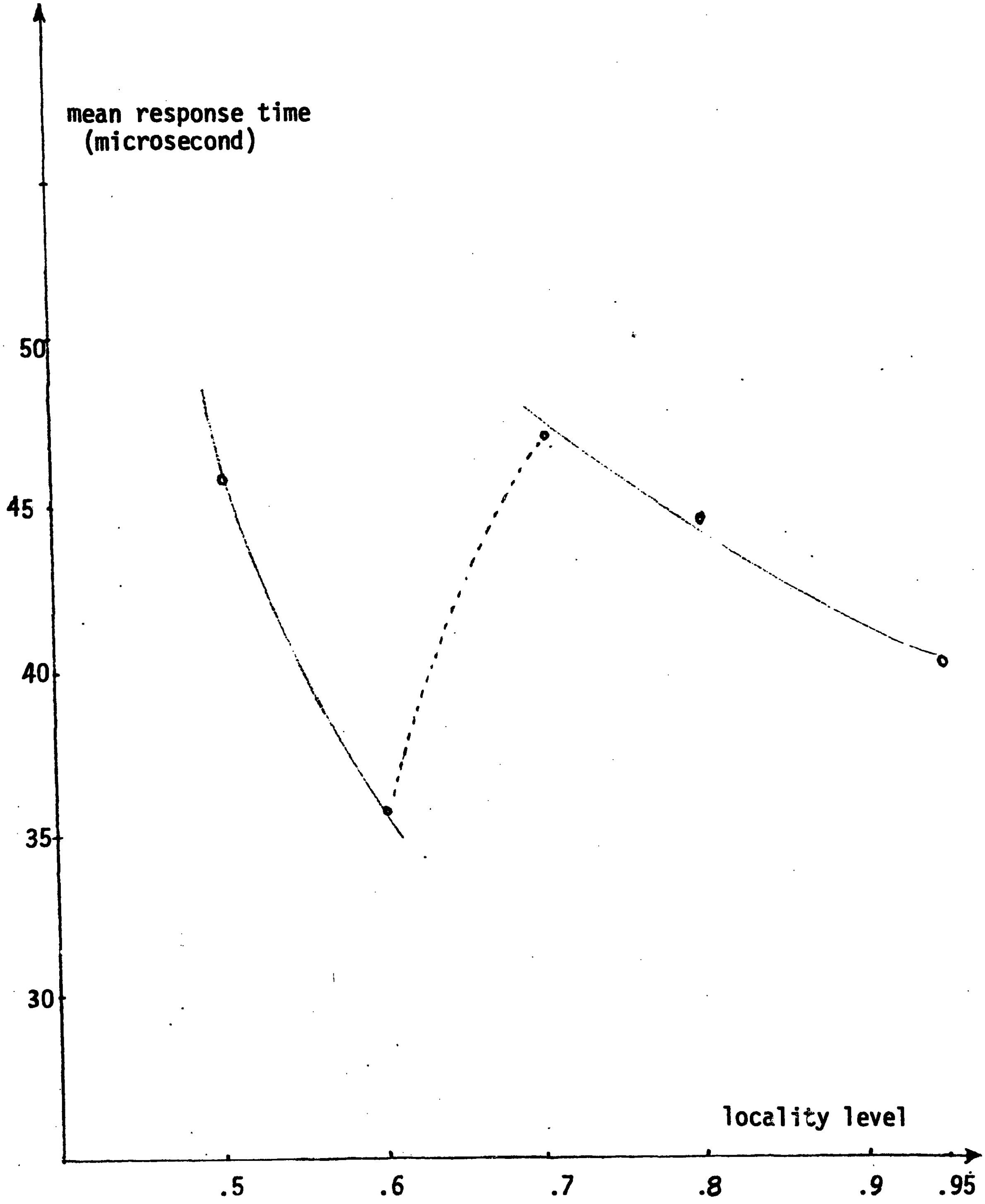


Figure 6.17 Mean Response Time Vs. Locality Level
(P5L4 Preliminary Results)

low. Figure 6.15(b) shows that the devices and the local bus at level 4 are satuarated for all locality levels. The local bus at level 3 is saturated but the devices at level 3 are only 50 percent utilized. Saturation of level 4 at low locality levels is due to the large number of read-through requests that has to be handled at that level. For example, at a locality level of .5, one-fouth of all read requests will be serviced by level 4. This creates a heavy burden on the level 4 devices and on its bus. At high locality levels, however, the number of read-through requests directed to level 4 is rather small. For example, at a locality level of .8, only .8 percent of all read requests are serviced by level 4. The saturation of level 4 at high locality levels is due to the store-behind requests. At high locality levels, the number of write requests are much higher, thus there is a high demand on level 4 to service the corresponding store-behind requests. It seems that level 3 storage devices have the capacity to handle the read-through and store-behind requests at all locality levels. However, the local bus at level 3 is saturated at all locality levels. The bus saturation at level 3 is possibly due to the store-behind requests. We shall discuss ways to eliminate these performance bottlenecks in a later section.

Throughput data presented in Figure 6.15(a) is plotted as a graph in Figure 6.16. Figure 6.16 shows that throughput rises sharply starting from the .5 locality level, then its follows a much slower rate of increase after the .7 locality level. At a low locality level, throughput is low since a large proportion of the read requests has to go to the slower storage devices. As the locality level increases, a large proportion of requests can be handled by the higher storage levels. The higher storage levels are not heavily utilized, thus they can complete the requests quickly. The faster transactions can be completed, the faster new transactions can arrive since the model is a closed one. This explains the sharp increase in throughput between .5 and .7 locality levels.

When the locality level is high, the rate of store-behind transactions coming into the model becomes high. Since there is a fixed proportion of reads and writes in the request stream, the throughput at high locality levels becomes limited by how fast the store-behind requests can be serviced. Thus, at high locality levels, increasing the locality level further will not produce a dramatic increase in throughput.

The plot of mean response time in Figure 6.17 provides further insights to the store-behind operations. Figure

6.17 shows that there is a discontinuity in the mean response time curve between .6 and .7 locality levels. The discontinuity may be explained as follows. As the locality level increases, the rate of store-behind transactions coming into the model also increases. Read operations become a less dominant factor of system performance. There is a pipeline of buffer slots for store-behind transactions. A write request is completed as soon as it has completed a write to its data cache and has placed a store-behind transaction in the store-behind pipeline. The store-behind transaction flows along the pipeline until it is serviced and terminated by a level 4 storage device. If a write request cannot find a slot in the store-behind pipeline, it has to wait. At high locality levels, the store-behind pipeline is full, hence, write operations tend to incure a larger wait time waiting for pipeline slots. It seems that the store-behind pipeline is full after the .7 locality level, causing long wait times by transactions, hence larger mean response times for all locality levels higher than .7. The store-behind pipeline is rot full for all locality levels below .7. Thus transactions have smaller mean response time in these cases. This expains the difference in behavior of the two mean response time curves.

The data seems to support this theory. Outputs from the simulation runs shows that the pipeline is full for all locality levels greater than and equal to .7. The total transaction time column in Figure 6.15(a) shows that there is a dramatic increase in the transaction wait time for all cases with locality level above .7. The figure also shows that the transaction wait time is a dominant portion of the total transaction time. Since mean response time is the ratio of total transaction time to total number of completed transactions, the more than doubling of the wait time going from .6 to .7 locality level is the key factor in the sudden increase in mean response time. The sudden increase in wait time is due to the fact that the pipeline is just filled up, new transactions begin to experience prolonged delays. These preliminary studies have provided valuable insights to the dynamics of the store-behind operation. We now have gained enough understanding of the model to tune it for better performance.

6.5.3 Tuning the P5L4 Model

Our objective in this next series of studies is to try to obtain a well-balanced system. From the preliminary studies, we know that to reduce mean response time we have to increase the efficiency of the store-behind pipeline. One approach to increase the efficiency of the pipeline is to increase the parallelism of the lower storage levels, so that the service times of the stages of the pipeline are better balanced. The preliminary studies also reveal that our initial choice of block sizes may not be appropriate for the system.

The approach that is taken to obtain a well-balanced system is as follows. The locality level is fixed at .9. Then the degree of parallelism in level 3 is increased by a factor of 5 and that of level 4 is increased by a factor of 1Ø. This is accomplished by decreasing the effective service times of the devices at these levels appropriately. Finally, the model is run for several choices of block sizes for the storage levels. The simulated time for these runs are much longer than in the preliminary studies to ensure that steady state behavior is observed. The results obtained are summarized in Figure 6.18.

The first study uses the same block sizes as those used in the preliminary studies. The results of this study are summarized in column one which clearly shows that level 4 is the bottleneck causing the very low throughput and high mean response time. Note that the devices are not saturated. This indicates that the block sizes are too large thus tieing up the bus at level 4 during data transfer.

locality level = .90			
block sizes (bytes)	(8,128,1024)	(8,64,512)	(8,64,256)
throughput (per ms)	176	458	721
MRESP TIME (ns)	258580	96260	60940
gbus util.	.62	.67	.77
lbus1 util.	.02	.04	.07
lbus2 "	.10	.15	.26
lbus3 "	.67	.71	.84
lbus4 "	1.00	.99	.99
L1 "	.01	.04	.06
L2 "	.03	.07	.11
L3 "	.28	.27	.28
L4 "	.17	.40	.83

Figure 6.18 Tuning the P5L4 Model

In the next study, the size of data transfer between level 2 and level 3 and that between level 3 and level 4 are reduced by one half. The results of this study are summarized in column 2. The bus at level 4 is still a bottleneck. There is significant improvement in the utilizations of level 4 storage devices.

Next, the size of data transfer between level 3 and level 4 is halved. This produces a fairly well-balanced system. The results are summarized in column 3. A throughput of .7 million operations per second with mean response time of 60 microseconds is obtained. The utilizations across storage levels are well-balanced comparatively.

6.5.4 Comparing the Performance of DSH-11 using 1979 and 1985 Technologies

The well-balanced system obtained from the previous studies will be used as a basis for comparing the performance of DSH-11 under 1979 and 1985 technology assumptions. The parameters used in the 1979 case are exactly those used in the well-balanced system of the previous studies. For the 1985 case, we will use a bus speed that is 5 times faster than that used in the 1979 case. In general, the speeds of the storage devices in the 1985 case will be faster. We estimate that the level 1 storage devices will be twice as fast in 1985 as in 1979. All other devices are estimated to be 1Ø times faster in 1985 than in 1979. Lastly, we expect 1985 to produce better associative processors for directory searching thus the directory search time will be reduced by one half in 1985. These estimates will be incorproated in the parameters for the 1985 case.

The model using 1979 technology assumptions is run for 4 different request streams with different proportions of reads and writes. The model using 1985 technology assumptions is then run with the same 4 different request streams. The locality level is fixed at .9 in both cases. The results are summarized in Figure 6.19.

The throughputs for the two cases are plotted on the same graph in Figure 6.2Ø. In general, for both cases, throughput increases as the proportion of read requests increases. It can be inferred from the results that the throughput of DSH-11 using 1985 technology is between 5 to 1Ø times better than using 1979 technology. For a request stream with 7Ø percent read requests and 3Ø percent write requests, DSH-11 using 1979 technology can support a throughput of .7 million requests per second with a mean response time of 6Ø microseconds. For the same mix of requests, DSH-11 using 1985 technology can support a throughput of 4 million requests per second with a mean response time of 1Ø microseconds.

% read	TRUPT (per ms)	MRESP (ns)	gbus U.	lbus1 U.	lbus2 U.	lbus3 U.	lbus4 U.	L1 U.	L2 U.	L3 U.	L4 U.
.50	450	97580	.76	.06	.25	.84	.99	.04	.10	.25	.67
.70	721	60940	.77	.07	.26	.84	.99	.06	.11	.28	.65
.80	1559	26790	.85	.10	.34	.91	.97	.11	.18	.34	.71
.90	3239	13440	.90	.14	.42	.93	.97	.23	.28	.35	.83

(a) 1979 Technology

(b) 1985 Technology

%read	TRUPT (per ms)	MRESP (ns)	gbus U.	lbus1 U.	lbus2 U.	lbus3 U.	lbus4 U.	L1 U.	L2 U.	L3 U.	L4 U.
.50	2298	19780	.76	.06	.24	.82	.99	.13	.05	.27	.35
.70	4320	9940	.79	.07	.28	.86	.98	.20	.06	.28	.34
.80	15040	2640	.96	.15	.47	.97	.92	.64	.14	.38	.28
.90	22760	1760	.95	.16	.47	.96	.91	.99	.17	.27	.34

Figure 6.19 Comparing Performance of P5L4
Using Different Technologies

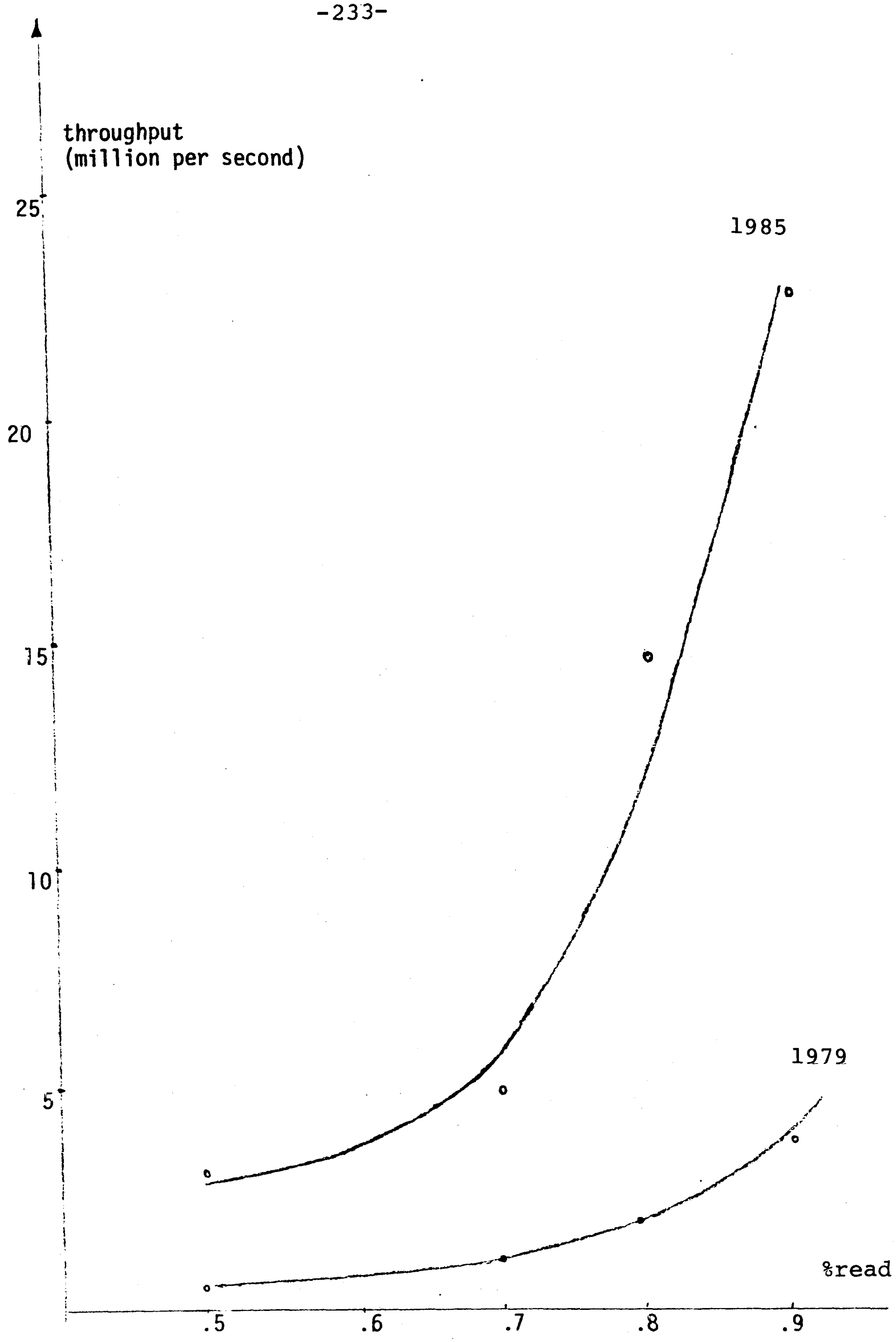


Figure 6.20 Throughput Vs. % Read
(Comparative Performace)

6.6 SUMMARY

Two simulation models of the DSH-11 storage hierarchy system are developed and used to understand the performance characteristics of DSH-11 and its algorithms. The first model is developed for a DSH-11 configuration with one processor and three storage levels. Results from this model uncovers an unsuspected deadlock potential in the DSH-11 buffer management scheme. This leads to the development of new buffer management schemes for DSH-11. A second model is developed for a DSH-11 configuration with five processors and four storage levels. This model also makes use of a deadlock-free buffer management scheme. Results from this model provides much insights to the performance implications of the read-through and store-behind algorithms. After sufficient understanding of the model is obtained, the model is tuned for better performance. The resulting system is then used as a basis for comparing the performance implication of using different technology for DSH-11.

Results from these simulation studies not only provide valuable insights to the important dynamic behavior of store-behind and read-through algorithms, they also provide assurance that the DSH-11 is capable of supporting the memory requirements of the INFOPLEX functional hierarchy.

Chapter VII

DISCUSSIONS AND CONCLUSIONS

7.1 INTRODUCTION

Database management is a major component of computer usage. Adapting conventional computer architectures to support database management functions has several disadvantages. Two major disadvantages have been recognized for some time. These are : (1) processor power limitation of the conventional computer, and (2) the 'access gap' that exists between main memory and secondary storage devices of conventional computers.

Various approaches have been proposed to develop specialized architectures for database management. These approaches have been discussed in Chapter 1. One of these approaches is the INFOPLEX data base computer effort. INFOPLEX eliminates the processor power limitation by using multiple specialized functional processors and makes use of a generalized storage hierarchy specifically designed for managing very large databases. A major obstacle to realize effective storage hierarchy systems has been the lack of understanding of these systems and their algorithms. Previous studies of storage hierarchy systems have been focused on systems with two or three storage levels, and usually for program storage. This thesis is focused on the study of generalized storage hierarchy systems for data storage, referred to as data storage hierarchy systems. Theories and models of data storage hierarchy systems are developed. Formal definitions of data management algorithms for data storage hierarchy systems are defined. Important properties of data storage hierarchy systems have been analyzed in detail to provide valuable insight for design of practical data storage hierarchy systems. Designs for the INFOPLEX data storage hierarchy are developed and protocols for realizing the read and write operations are specified. Finally, simulation models for these designs are developed to assess the feasibility of these designs for supporting the very high transaction rates of INFOPLEX and to obtain better understanding of the read-through and store-behind operations from a practical point of view.

7.2 SUMMARY OF THESIS

Chapter 1 of the thesis provides a framework for understanding the rationale behind various approaches to develop specialized machines for data management. Major contributions of this thesis are also listed.

The background and motivation for this research is the INFOPLEX data base computer project. Concepts of the INFOPLEX data base computer are presented in Chapter 2. An example functional hierarchy and an example storage hierarchy for INFOPLEX are used to illustrate some of these concepts.

A preliminary design of the INFOPLEX data storage hierarchy is proposed in Chapter 3. Design objectives and the structure of the system are presented. Further design issues that need to be resolved are also identified.

Formal modelling and analysis of data storage hierarchy systems are presented in Chapter 4. It contains formal proofs of the multi-level inclusion (MLI), the multi-level overflow inclusion (MLOI), and multi-level paging anomaly (MLPA) properties of data storage hierarchy systems.

The preliminary design of the INFOPLEX data storage hierarchy system presented in Chapter 2 is simplified in Chapter 5. This simplified design is then used to develop protocols for supporting the read and write operations. Specifications for these protocols are presented in Chapter 5.

A simulation model of the INFOPLEX data storage hierarchy system with one functional processor and three storage levels is developed in Chapter 6. Results from this simulation model are analyzed. Insights from these analysis lead to some design changes. Another simulation model of the INFOPLEX data storage hierarchy is then developed. This model incorporates five functional processors and four storage levels. Results from this model are analyzed and reveal further interesting properties of the design and of the data management algorithms. The impacts of using projected 1985 technology are also assessed.

7.3 DIRECTIONS FOR FURTHER RESEARCH

This thesis has provided a theoretic framework for formal analysis of data storage hierarchy systems. Using this framework, several important properties of data storage hierarchy systems that have performance and reliability implications are studied in detail. This work also opens up many areas for further investigation. Do the properties of data storage hierarchy systems proved in this thesis hold for systems using any stack algorithm (Mattson et. al., 1970)? What are the effects of introducing the two-level store-behind algorithm into the system? Are the conditions for avoiding the multi-level paging anomaly (MLPA) also necessary conditions, i.e., what are the causes of MLPA? These are interesting and important questions. The formal basis developed in this thesis will be a major steping stone toward resolving these open questions.

The preliminary design of the INFOPLEX data storage hierarchy can be used to develop algorithms that improve the efficiency and reliability of the data storage hierarchy. The automatic data repair algorithms introduced in Chapter 3 are particularly interesting and promising. A number of other design issues are discussed but left as open issues. For example, the multi-cache consistency problem by itself is a subject of great importance but still quite lacking of theoretic basis for analysis.

The simulation results reveal several important properties of the design and of the algorithms that are quite unexpected. The deadlock potential in the initial design can be corrected quite easily. The fact that the store-behind operation can be a system bottleneck is not anticipated before. It has been argued in the past that store-behind operations take place during system slack periods thus do not adversly impact system performance. A number of alternative schemes can be developed to improve the efficiency of the store-behind operation. May be we can separate the read only data from the read/write data and keep the read/write data higher up in the data storage hierarchy system. This would reduce the store-behind traffic to lower storage levels. The implications of this type of data management strategy remain uncharted.

Some of these issues are currently being addressed as part of the INFOPLEX research effort (Abraham, 1979).

REFERENCES

(Abe, 1977) : Abe, Y., 'A Japanese On-line Banking System', Datamation, September 1977, 89-97.

(Abraham, 1979): Abraham, M. J., ' Properties of Reference Algorithms for Multi-level Storage Hierarchies', Masters Thesis, M.I.T. Sloan School of Management, Cambridge, Mass.,June 1979.

(Ahearn et. al., 1972) : Ahearn, G.P., Dishon, Y., and Snively, R.N., 'Design Innovations of the IBM 3830 and 2835 Storage Control Units', IBM Journal of Research and Development 16, 1 (January, 1972), 11-18.

(ANSI, 1975) : ANSI, 'Interim Report of ANSI/X3/SPARC group on Database Management Systems', ANSI, February, 1975.

(Armenti et. al., 1970) : Armenti, A., Galley, S., Goldberg, R., Nolan, J., and Scholl, A., 'LISTAR - Lincoln Information Storage and Associatve Retrieval System', AFIP Conference Proceedings, 36, (1970), 313-322.

(Arora and Gallo, 1971) : Arora, S.R., and Gallo, A., 'Optimal Sizing Loading and Reloading in a Multi-level Memory Hierarchy System', Spring Joint Computer Conference, 1971, 337-344.

(Bachman, 1975) : Bachman, C., 'Trends in Database Management - 1975', AFIP Conference Proceedings, 44, (1975), 569-576.

(Banerjee et. al., 1978) : Banerjee, J., Hsiao, D.K., and Baum, R.I., 'Concepts and Capabilities of a Database Computer', ACM Trans. on Database Systems, 3, 4 (December, 1978), 347-384.

(Banerjee et. al., 1979) : Banerjee, J., Hsiao, D.K., and Kannan, K., 'DBC - A Database Computer for Very Large Databases', IEEE Trans. on Computers, C-28, 6 (June 1979), 414-429.

(Belady, 1966) : Belady, L.A., 'A Study of Replacement Algorithms for a Virtual-storage Computer', IBM Systems Journal, 5, 2, (1966), 78-101.

(Belady *et. al.*, 1969) : Belady, L.A., Nelson, R.A., and Shedler, G.S., 'An Anomaly in Space-time Characteristics of Certain Programs Running in a Paging Machine', Comm. ACM, 12, 6, (June, 1969), 349-353.

(Bensoussan *et. al.*, 1969): Bensoussan, A., Clingen, C.T., and Daley, R.C., 'The MULTICS Virtual Memory', Second Symposium on Operating Systems Principles, Princeton University, October 1969, 30-42.

(Canaday *et. al.*, 1974) : Canaday, R.H., Harrison, R.D., Ivie, E.L., Ryder, J.L., and Wehr, L.A., 'A Back-end Computer for Database Management', Comm. ACM, 17,10 (October 1974), 575-584.

(Censier and Feautrier, 1978) : Censier, L.M., and Feautrier, P., 'A New Solution to Coherence Problems in Multicache Systems', IEEE Trans. on Computers, C-27, 12 (December, 1978), 1112-1118.

(Chen, 1973) : Chen, P.P., 'Optimal File Allocation in Multi-level Storage Systems', Proceedings National Computer Conference, June 1973.

(Codasyl, 1971) : Codasyl, 'Data Base Task Group, April 1971 Report', ACM, New York, 1971.

(Codd, 1970) : Codd, E.F., 'A Relational Model of Data for Large Shared Data Banks', Comm. ACM, 13, 6 (June 1970), 377-387.

(Codd, 1974) : Codd, E.F., 'Recent Investigations in Relational Database Systems', Information Processing 71, North Holland Publishing Company, 1974.

(Computerworld, 1976) : Computerworld, 'Reliability Seen Key to TWA Reservations System', Computerworld, September 6, 1976, 22.

(Conti, 1969) : Conti, C.J., 'Concepts for Buffer Storage', IEEE Computer Group News, March 1969, 6-13.

(Considine and Weis, 1969): Considine, J.P., and Weis, A.H., 'Establishment and Maintenance of a Storage Hierarchy for an Online Database Under TSS/360', Fall Joint Computer Conference, 35 (1969), 433-440.

(Copeland et. al., 1973) : Copeland, G.P., Lipovski, G.J., and Su, S.Y.W., 'The Architecture of CASSM: A Cellular System for Non-numeric Processing', Proceedings of First Annual Symposium on Computer Architecture, December, 1973, 121-128.

(Datamation, 1978) : Datamation, December, 1978, 230.

(Denning, 1970) : Denning, P.J., 'Virtual Memory', ACM Computing Surveys, 2, 3 (September 1970), 153-190.

(Dennis et. al., 1978) : Dennis, J.B., Fuller, S.H., Ackerman, W.B., Swan, R.J., and Weng, K., 'Research Directions in Computer Architecture', M.I.T. Laboratory for Computer Sciences, LCS-TM-114, September, 1978.

(Dijkstra, 1968) : Dijkstra, E.W., 'The Structure of T.H.E. Multiprogramming System', Comm. ACM, 11, 5 (May 1968).

(Donovan and Jacoby, 1975) : Donovan, J.J., and Jacoby, H.D., 'A Hierarchical Approach to Information System Design', CISR-5, Sloan School of Management, MIT, January 1975.

(Easton, 1978): Easton, M.C., 'Model for Database Reference Strings Based on Behavior of Reference Clusters', IBM Journal of Research and Development, 22, 2 (March, 1978), 197-202.

(Folinus et. al., 1974) : Folinus, J.J., Madnick, S.E., and Schutzman, H., 'Virtual Information in Database Systems', Working Paper CISR-3, Sloan School of Management, MIT, July 1974.

(Franaszek and Bennett, 1978) : Franaszek, P.A., and Bennett, B.T., 'Adaptive Variation of the Transfer Unit in a Storage Hierarchy', IBM Journal of Research and Development, 22, 4, (July, 1978), 405-412.

(Franklin et. al., 1978) : Franklin, M.A., Graham, G.S., and Gupta, R.K., 'Anomalies with Variable Partition Paging Algorithms', Comm. ACM, 21, 3, (March, 1978), 232-236.

(Goldberg, 1974) : Goldberg, R.P., 'The Double Paging Anomaly', Proceedings National Computer Conference, 1974, 195-199.

(Greenberg and Webber, 1975) : Greenberg, B.S., and Webber, S.H., 'MULTICS Multilevel Paging Hierarchy', IEEE INTERCON, 1975.

(Haagens, 1978) : Haagens, R.B., 'A Bus Structure for Multi-Microprocessing', Masters Thesis, M.I.T. Department of Electrical Engineering, Cambridge, Mass., January, 1978.

(Hakozaki et. al., 1977) : Hakozaki, K., Makino, T., Mizuma, M., Umemura, M., and Hiyoshi, S., 'A Conceptual Design of a Generalized Database Subsystem', Proc. Very Large Data Bases, 1977, 246-253.

(Hatfield, 1972) : Hatfield, D.J., 'Experiments on Page Size, Program Access Patterns, and Virtual Memory Performance', IBM Journal of Research and Development, 16, 1 (January 1972), 58-66.

(Hatfield and Gerald, 1971) : Hatfield, D.J., and Gerald, J., 'Program Restructuring for Virtual Memory', IBM Systems Journal, 10, 3, (1971), 168-192.

(Healy et. al., 1972) : Healy, L.D., Doty, K.L., and Lipovski, G.J., 'The Architecture of a Context Addressed Segment Sequential Storage', AFIPS Conference Proceedings, 41, (1972), 691-701.

(Hoagland, 1979) : Hoagland, A.S., 'Storage Technology: Capabilities and Limitations', Spring COMPCON, 1979, 60-64.

(Hsiao and Kannan, 1976) : Hsiao, D.K., and Kannan, K., 'The Architecture of a Database Computer - Part II: The Design of Structure Memory and its Related Processors', OSU-CISRC-TR-76-e, Ohio State University, December 1976.

(Huff and Madnick, 1978) : Huff, S.L., and Madnick, S.E., 'An Approach to Constructing Functional Requirement Statements for System Architectural Design', M.I.T. Sloan School of Management, C.I.S.R. Internal Report No. P010-7806-06, (June 1978).

(Johnson, 1975) : Johnson, C., 'IBM 3850 - Mass Storage System', AFIPS Conference Proceedings, 44, (1975), 509-514.

(Johnson J, 1975) : Johnson, J., 'Program Restructuring for Virtual Memory Systems', MIT Project MAC, TR-148, March, 1975.

(Katsuki et. al., 1978) : Katsuki, D., Elsam, E.S., Mann, W.F., Roberts, E.S., Robinson, J.G., Skowronski, F.S., and Wolf, E.W., 'Pluribus - An Operational Fault-Tolerant Multiprocessor', Proceedings of the IEEE, 66, 10 (October 1978), 1146-1159.

(Lam and Madnick, 1979) : Lam, C.Y., and Madnick, S.E., 'INFOPLEX Data Base Computer Architecture - Concepts and Directions', MIT Sloan School Working Paper No. 1046-79 (also as CISR Working Paper No. 41), 1979.

(Lang et. al., 1977) : Lang, T., Nahouraii, E. Kasuga, K. and Fernadez, E.B., 'An Architectural Extension for a Large Database System Incorporating a Processor for Disk Search', Proc. Very Large Data Bases, 1977, 204-210.

(Langdon, 1978) : Langdon G.G. Jr., 'A Note on Associative Processing for Data Management', ACM Trans. on Database Systems, 3, 2 (June 1978), 148-158.

(Lin et. al., 1976) : Lin, S.C., Smith, D.C.P., and Smith, J.M. 'The Design of a Rotating Associative Memory for Relational Database Applications'. ACM Trans. on Database Systems, 1, 1 (March 1976), 53-75.

(Lum et. al., 1975) : Lum, V.Y., Senko, M.E., Wang, C.P., and Ling, H., 'A Cost Oriented Algorithm for Data Set Allocation in Storage Hierarchies', Comm. ACM, 18, 6, (June, 1975), 318-322.

(Madnick, 1970) : Madnick, S.E., 'Design Strategies for File Systems', MIT Project MAC Report No. TR-78, October 1970.

(Madnick, 1973) : Madnick, S.E., 'Storage Hierarchy Systems', MIT Project MAC Report No. TR-105, 1973.

(Madnick, 1975a) : Madnick, S.E., 'Design of a General Hierarchical Storage System', IEEE INTERCON Proceedings, 1975, 1-7.

(Madnick, 1975b) : Madnick, S.E., 'INFOPLEX - Hierarchical Decomposition of a Large Information Management System Using a Microprocessor Complex', Proceedings National Computer Conference, 44, (May 1975), 581-587.

(Madnick, 1977) : Madnick, S.E., 'Trends in Computers and Computing: The Information Utility', Science, 195, 4283 (1973), 1191-1199.

(Madnick, 1979): Madnick, S.E., 'The INFOPLEX Database Computer: Concepts and Directions', Proceedings IEEE Computer Conference, February 26, 1979, 168-176.

(Madnick and Alsop, 1969) : Madnick, S.E., and Alsop, J. 'A Modular Approach to File System Design', Spring Joint Computer Conference Proceedings, 34 (May 1969), 1-14.

(Madnick and Donovan, 1974) : Madnick, S.E., and Donovan, J.J., Operating Systems, McGraw-Hill, New York, 1974.

(Marill and Stern, 1975) : Marill, T., and Stern, D., 'The Datacomputer - a Network Data Utility', AFIPS Conference Proceedings, 44 (975), 389-395.

(Martin, 1975) : Martin, J., Computer Data Base Organization, Prentice-Hall, New York, 1975.

(Mattson, 1971) : Mattson, R.L., 'Evaluation of Multilevel Memories', IEEE Transactions on Magnetics, Vol. MAG-7, No. 4 (DEcember, 1971), 814-819.

(Mattson et. al., 1970) : Mattson, R.L., Gecsei, J., Slutz, D.R., and Traiger, I.L., 'Evaluation Techniques for Storage Hierarchies', IBM Systems Journal, 9, 2 (1970), 78-117.

(McCabe, 1978): McCabe, E.J., 'Locality in Logical Database Systems: A Framework for Analysis', Masters Thesis, M.I.T. Sloan School of Management, Cambridge, Mass., July 1978.

(Meade, 1970): Meade, R.M., 'On Memory System Design', AFIPS Conference Proceedings, 37 (1970), 33-43.

(Mohan, 1978) : Mohan, C., 'An Overview of Recent Data Base Research', DATABASE, Fall 1978, 3-24.

(Mueller, 1976) : Mueller, G., Computer, 9, 12 (December 1976), 100.

(Ozkarahan et. al., 1975) : Ozkarahan, E.A., Schuster, S.A., and Smith, K.C., 'RAP - Associative Processor for Data Base Management', AFIPS Conference Proceedings, 44, (1975), 379-388.

(Ozkarahan et. al., 1977) : Ozkarahan, E.A., Schuster, S.A., and Sevcik, K.C., 'Performance Evaluation of a Relational Associative Processor', ACM Trans. on Database Systems, 2, 2 (June, 1977), 175-195.

(Parnas, 1976) : Parnas, D.L., 'On The Design and Development of Program Families', IEEE Transactions on Software Engineering, SE-2-1, March 1976.

(Pattee, 1973) : Pattee, H.H., *Hierarch Theory: The Challenge of Complex Systems*, George Brazillier, New York, 1973.

(Ramamoothy and Chandy, 1970) : Ramamoorthy, C.V., and Chandy, K.M., 'Optimization of Memory Hierarchies in Multiprogrammed Systems', Journal of the ACM, July 1970.

(Robidoux, 1979): Robidoux, S.L., 'A Closer Look at Database Access Patterns', Masters Thesis, M.I.T. Sloan School of Management, Cambridge, Mass., June, 1979.

(Rodriguez-Rosell, 1976): Rodriguez-Rosell, J., 'Empirical Data Reference Behavior in Data Base Systems', Computer, November, 1976, 9-13.

(Scherr, 1973) : Scherr, A.L., 'Functional Structure of IBM Virtual Storage Operating Systems Part II : OS/VS2-2 Concepts and Philosophies', IBM Systems Journal, 12, 4 (1973), 382-400.

(Schuster, 1978) : Schuster, S.A., 'Data Base Machines', Proceedings of the Conference on Computing in the 1980's, 1978, 125-131.

(Schuster *et*. *al*., 1976) : Schuster, S.A., Ozkarahan, E.A., and Smith, K.C., 'A Virtual Memory System for a Relational Associative Processor', Proceedings National Computer Conference, 1976, 855-862.

(Schuster *et*. *al*., 1979) : Schuster, S.A., Nguyen, H.B., Ozkarahan, E.A., and Smith, K.C., 'RAP.2 - An Associative Processor for Databases and Its Applications', IEEE Trans. on Computers, C-28, 6 (June 1979), 446-458.

(Senko, 1976) : Senko, M.E., 'DIAM II: The Binary Infological Level and its Database Language - FORAL', Proceedings of the ACM Conference on Data. March 1976, 1-21.

(Simonson and Alsbrooks, 1975) : Simonson, W.E., and Alsbrooks, W.T., 'A DBMS for the U.S. Bureau of the Census', Proc. Very Large Data Bases, 1975, 496-497.

(Smith, 1978a) : Smith, A.J., 'Directions for Memory Hierarchies and their Components: Research and Development', IEEE Proceedings COMPSAC, 1978, Chicago, 704-709.

(Smith, 1978b): Smith, A.J., 'Sequentiality and Prefetching in Data Base Systems', ACM Trans. on Database Systems, September, 1978.

(Soltis and Hoffman, 1979) : Soltis, F.G., and Hoffman, R.L., 'Design Considerations for the IBM System/38', Spring COMPCON, 1979, 132-137.

(Su, 1977) : Su, S.Y.W., 'Associative Programming in CASSM and its Applications', Proc. Very Large Data Bases, 1977, 213-228.

(Su and Lipovski, 1975) : Su, S.Y., and Lipovski, G.J., 'CASSM: A Cellular System for Very Large Databases', Proc. Very Large Data Bases, September 1975, 456-472.

(Su et. al., 1979) : Su, S.Y.W., Nguyen, L.H., Emam, A., and Lipovski, G.J., 'The Architectural Features and Implementation Techniques of the Multicell CASSM', IEEE Trans. on Computers, C-28, 6 (June 1979), 430-445.

(Tang,1976) : Tang, C.K., 'Cache System Design in the Tightly Coupled Multiprocessor System', Proceedings National Computer Conference, 1976, 749-753.

(Toh et. al., 1977) : Toh, T., Kawazu, S., and Suzuki, K., 'Multi-Level Structures of the DBTG Data Model for an Achievement of the Physical Independence', Proc. Very Large Data Bases, 1977, 403-414.

(Yeh et. al., 1977) : Yeh, R.T., and Baker, J.W., 'Toward a Design Methodology for DBMS: A Software Engineering Approach', Proc. Very Large Data Bases, 1977, 16-27.

Appendix A

LISTING OF THE P1L3 MODEL

```
//LAM1 JOB LAM,MPROFILE='RETURN',
// PROFILE='HIGH',
// TIME=2
//*PASSWORD
//GPSS PROC
//C   EXEC  PGM=DAG01,TIME=2
//STEPLIB DD DSN=POTLUCK.LIBRARY.GPSS.LOAD,DISP=SHR
//DOUTPUT DD SYSOUT=PROFILE=RETURN,DCB=BLKSIZE=931
//DINTERO DD UNIT=SCRATCH,SPACE=(CYL,(1,1)),DCB=BLKSIZE=1880
//DSYMTAB DD UNIT=SCRATCH,SPACE=(CYL,(1,1)),DCB=BLKSIZE=7112
//DREPTGEN DD UNIT=SCRATCH,SPACE=(CYL,(1,1)),DCB=BLKSIZE=800
//DINTWORK DD UNIT=SCRATCH,SPACE=(CYL,(1,1)),DCB=BLKSIZE=2680
// PEND
//STEP1  EXEC  GPSS,PARM=C
//DINPUT1  DD *
*****************************************************************
*                                                               *
*     TRANSACTION PARAMETER USAGE                               *
*                                                               *
*     P1     CPU IDENTIFIER                                     *
*     P2     ARRIVAL TIME                                       *
*     P3     COMPLETION TIME                                    *
*     P4     TOTAL EXECUTION TIME                               *
*     P5     TOTAL ELAPSED TIME                                 *
*     P6     TOTAL WAIT TIME                                    *
*     P7     SERVICE TIME                                        *
*     P11    DUMMY                                              *
*                                                               *
*****************************************************************
*
 NTXN  EQU        01,X        NUMBER OF TXNS PROCESSED
 SUMX  EQU        02,X        EXECUTION TIME OF ALL TXNS
 SUMQ  EQU        03,X        QUEUE TIME OF ALL TXNS
 SUMW  EQU        04,X        ELAPSED TIME OF ALL TXNS
*
 MAXMP EQU        05,X        DEGREE OF CPU MULTIPLROGRAMMING
 NREAD EQU        06,X        PARTS IN THOUSAND OF READ TXNS
 NWRIT EQU        07,X        PARTS IN THOUSAND OF WRITE TXNS
*
 PIN1  EQU        08,X        PROB OF FINDING READ DATA IN L(1)
 PIN2  EQU        09,X        PROB OF FINDING READ DATA IN L(2)
 PIN3  EQU        10,X        PROB OF FINDING READ DATA IN L(3)
*
 POV1  EQU        11,X        PROB OF OVERFLOW FROM L(1)
 POV2  EQU        12,X        PROB OF OVERFLOW FROM L(2)
 POV3  EQU        13,X        PROB OF OVERFLOW FROM L(3)
*
*****************************************************************
*        MAXIMUM DATA QUEUE LENGTHS                             *
*****************************************************************
*
 DXM11 EQU        14,X
 DYM11 EQU        15,X
 DXM12 EQU        16,X
 DYM12 EQU        17,X
 DXM13 EQU        18,X
 DYM13 EQU        19,X
*
 DXM21 EQU        20,X
 DYM21 EQU        21,X
 DXM22 EQU        22,X
 DYM22 EQU        23,X
*
 DXM31 EQU        24,X
 DYM31 EQU        25,X
 DXM32 EQU        26,X
 DYM32 EQU        27,X
*
 KXM1  EQU        28,X
 KYM1  EQU        29,X
*
 KXM2  EQU        30,X
 KYM2  EQU        31,X
*
 KXM3  EQU        32,X
 KYM3  EQU        33,X
*
 RXM2  EQU        34,X
 RYM2  EQU        35,X
*
 RXM3  EQU        36,X
 RYM3  EQU        37,X
*
*****************************************************************
*     CURRENT LENGTHS OF DATA QUEUES                            *
*****************************************************************
*
 DXL11 EQU        38,X
 DYL11 EQU        39,X
 DXL12 EQU        40,X
 DYL12 EQU        41,X
 DXL13 EQU        42,X
 DYL13 EQU        43,X
*
 DXL21 EQU        44,X
 DYL21 EQU        45,X
 DXL22 EQU        46,X
 DYL22 EQU        47,X
*
 DXL31 EQU        48,X
 DYL31 EQU        49,X
 DXL32 EQU        50,X
 DYL32 EQU        51,X
*
 KXL1  EQU        52,X
 KYL1  EQU        53,X
*
 KXL2  EQU        54,X
 KYL2  EQU        55,X
*
```

```
 KXL3  EQU        56,X
 KYL3  EQU        57,X

 RXL2  EQU        58,X
 RYL2  EQU        59,X
*
 RXL3  EQU        60,X
 RYL3  EQU        61,X
*
****************************************************************
* SERVICE TIMES OF DEVICES, BUSES, PROCESSORS                  *
****************************************************************

 DEX11 EQU        62,X        L(1) STORAGE SERVICE TIME
 DEX12 EQU        63,X
 DEX13 EQU        64,X
 DEX21 EQU        65,X        L(2) STORAGE SERVICE TIMES
 DEX22 EQU        66,X
 DEX31 EQU        67,X        L(3) STORAGE SERVICE TIMES
 DEX32 EQU        68,X
 BEXD1 EQU        69,X        BUS SERV TIME L(1)
 BEXD2 EQU        70,X        BUS SERV. TIME L(2)
 BXD3  EQU        71,X        BUS SERV. TIME L(3)
 BEXM  EQU        72,X        BUS SERV. TIME FOR MSG
 KEX   EQU        73,X        LEVEL CONTROLLER (K) SERVICE TIME
 REX   EQU        74,X        MEMORY REQUEST PROCESSOR (R) SERVICE TIME
 TIMER EQU        75,X

****************************************************************
*                                                              *
*  VARAIBLE DEFINITIONS                                        *
*                                                              *
****************************************************************

 MRESP FVARIABLE  (X$SUMW/X$NTXN)              MEAN RESPONSE TIME
 TXNW  VARIABLE   P3-P2                        TXN ELAPSED TIME
 TXNQ  VARIABLE   P3-P2-P4                     TXN WAIT TIME
 TXNX  VARIABLE   P4
 RTOK  BVARIABLE  (X$KXL1'L'X$KXM1)*(X$KXL2'L'X$KXM2)*FNU$GBUS
 BVA1  BVARIABLE  (X$DYL11'L'X$DYM11)*FNU$DRP11
 BVA2  BVARIABLE  (X$KXL1'L'X$KXM1)*FNU$LBUS1
 BVA3  BVARIABLE  (X$DYL21'L'X$DYM21)*FNU$DRP21
 BVA21 BVARIABLE  (X$DYL22'L'X$DYM22)*FNU$DRP22
 BVA4  BVARIABLE  (X$KXL2'L'X$KXM2)*FNU$LBUS2
 BVA5  BVARIABLE  (X$KYL2'L'X$KYM2)*FNU$KRP2
 BVA6  BVARIABLE  (X$KXL1'L'X$KXM1)*FNU$GBUS
 BVA7  BVARIABLE  (X$DXL11'L'X$DXM11)*FNU$LBUS1
 BVA8  BVARIABLE  (X$DYL31'L'X$DYM31)*FNU$DRP31
 BVA22 BVARIABLE  (X$DYL32'L'X$DYM32)*FNU$DRP32
 BVA9  BVARIABLE  (X$KXL3'L'X$KXM3)*FNU$LBUS3
 BVA10 BVARIABLE  (X$KYL3'L'X$KYM3)*FNU$KRP3
 BVA11 BVARIABLE  (X$RXL2'L'X$RXM2)*FNU$LBUS2
 BVA12 BVARIABLE  (X$RYL2'L'X$RYM2)*FNU$RRP2
 BVA13 BVARIABLE  (X$DXL21'L'X$DXM21)*FNU$LBUS2
 BVA23 BVARIABLE  (X$DXL22'L'X$DXM22)*FNU$LBUS2
 BVA14 BVARIABLE  (X$KYL1'L'X$KYM1)*FNU$KRP1
 BVA15 BVARIABLE  (X$KXL2'L'X$KXM2)*FNU$GBUS
 BVA16 BVARIABLE  (X$KXL3'L'X$KXM3)*FNU$GBUS
 BVA17 BVARIABLE  (X$RXL3'L'X$RXM3)*FNU$LBUS3
 BVA18 BVARIABLE  (X$RYL3'L'X$RYM3)*FNU$RRP3
 BVA19 BVARIABLE  (X$DXL31'L'X$DXM31)*FNU$LBUS3
 BVA24 BVARIABLE  (X$DXL32'L'X$DXM32)*FNU$LBUS3
 BVA20 BVARIABLE  (X$KYL1'L'X$KYM1)*FNU$KRP1

****************************************************************
*                                                              *
*  QTABLE DEFINITIONS - DISTRIBUTIONS OF QUEUE LENGTHS         *
*                                                              *
****************************************************************

****************************************************************
*                                                              *
* FUNCTION DEFINITIONS                                         *
*                                                              *
****************************************************************

 WICHW FUNCTION   P1,D3
2,WWW11/3,WWW12/4,WWW13

 WICHA FUNCTION   P1,D3
2,AAA11/3,AAA12/4,AAA13


****************************************************************
*                                                              *
* TABLE DEFINITIONS - DISTRIBUTIONS OF TXN ELAPSED TIME,       *
*                     WAIT TIME                                *
*                                                              *
****************************************************************

 TXNW  TABLE      V$TXNW,100,100,100
 TXNQ  TABLE      V$TXNQ,100,100,100
 TXNX  TABLE      V$TXNX,100,100,100


****************************************************************
*                                                              *
*  INITIALIZE CONSTANTS                                        *
*                                                              *
****************************************************************

       INITIAL    X$MAXMP,20        DEGREE CF MULTIPROGRAMMING OF A CPU
       INITIAL    X$NREAD,700       % READ TXN
       INITIAL    X$NWRIT,300       % WRITE TXN
       INITIAL    X$PIN1,400        PROB OF FINDING READ DATA IN L(1)
       INITIAL    X$PIN2,400        PROB OF NCT IN L(1) AND IN L(2)
       INITIAL    X$PIN3,1000       PRCB OF FINDING DATA IN L(3)
       INITIAL    X$POV1,500        PROB OF OVERFLOW FROM L(1)
       INITIAL    X$POV2,500        PROB OF OVERFLOW FROM L(2)
       INITIAL    X$DXM11,10        MAXIMUM DATA QUEUE LENGTH
```

```
      INITIAL    X$DYM11,10
      INITIAL    X$DXM12,10
      INITIAL    X$DYM12,10
      INITIAL    X$DXM13,10
      INITIAL    X$DYM13,10
      INITIAL    X$DXM21,10
      INITIAL    X$DYM21,10
      INITIAL    X$DXM22,10
      INITIAL    X$DYM22,10
      INITIAL    X$DXM31,10
      INITIAL    X$DYM31,10
      INITIAL    X$DXM32,10
      INITIAL    X$DYM32,10
      INITIAL    X$KXM1,10
      INITIAL    X$KYM1,10
      INITIAL    X$KXM2,10
      INITIAL    X$KYM2,10
      INITIAL    X$KXM3,10
      INITIAL    X$KYM3,10
      INITIAL    X$RXM2,10
      INITIAL    X$RYM2,10
      INITIAL    X$RXM3,10
      INITIAL    X$RYM3,10
      INITIAL    X$DEX11,100       ACCESS TIME OF D11 IN NANOSEC
      INITIAL    X$DEX12,100
      INITIAL    X$DEX13,100
      INITIAL    X$DEX21,1000      ACCESS TIME OF D21 IN NANOSEC
      INITIAL    X$DEX22,1000
      INITIAL    X$DEX31,10000     ACCESS TIME OF D31 IN NANOSEC
      INITIAL    X$DEX32,10000
      INITIAL    X$BEXD1,100        BUS SERV. TIME IN NANOSEC
      INITIAL    X$BEXD2,1600
      INITIAL    X$BEXM,100
      INITIAL    X$KEX,100          L(I) CONTR. P. SERV. TIME IN NANOS
      INITIAL    X$REX,200          REQ. P. SERVICE TIME IN NANOS
      INITIAL    X$TIMER,1000000    SIMULATION TIME


************************************
*                                  *
* MACRO -UTX                       *
*                                  *
************************************

 UTX  STARTMACRO
      SEIZE       #A
      DEPART      #B
      ASSIGN      4+,#C
      ASSIGN      7,#C
      ADVANCE     P7
      RELEASE     #A
      ENDMACRO

************************************
```

```
*                                  *
*   MACRO - UQTQ                   *
*                                  *
************************************

 UQTQ   STARTMACRO
        QUEUE       #A
        SEIZE       #B
        DEPART      #A
        ASSIGN      4+,#D
        ASSIGN      7,#D
        ADVANCE     P7
        RELEASE     #B
        QUEUE       #C
        ENDMACRO

*************************
*                       *
*  MACRO - UQT          *
*                       *
*************************

 UQT    STARTMACRO
        QUEUE       #A
        SEIZE       #B
        DEPART      #A
        ASSIGN      4+,#C
        ASSIGN      7,#C
        ADVANCE     P7
        RELEASE     #B
        ENDMACRO

************************************
*                                  *
*  MACRO - UQDQ                    *
*                                  *
************************************

 UQDQ   STARTMACRO
        QUEUE       #A
        TEST E      #G,1
        SAVEVALUE   #D,1
        SEIZE       #E
        DEPART      #A
        SAVEVALUE   #B,1
        ASSIGN      4+,#F
        ASSIGN      7,#F
        ADVANCE     P7
        RELEASE     #E
        QUEUE       #C
        ENDMACRO

*************************
*                       *
*  MACRO - UQD          *
*                       *
*************************

 UQD    STARTMACRO
        QUEUE       #A
        TEST E      #G,1
        SAVEVALUE   #D,1
        SEIZE       #E
        DEPART      #A
        SAVEVALUE   #B,1
        ASSIGN      4+,#F
        ASSIGN      7,#F
        ADVANCE     P7
        RELEASE     #E
        ENDMACRO

***********************************
*                                 *
*   MACRO - FINI                  *
*                                 *
***********************************

 FINI   STARTMACRO
        MARK        3
        SAVEVALUE   NTXN+,1
        SAVEVALUE   SUMX+,V$TXNX
        SAVEVALUE   SUMQ+,V$TXNQ
        SAVEVALUE   SUMW+,V$TXNW
        SAVEVALUE   MRESP,V$MRESP
        TABULATE    TXNW
        TABULATE    TXNQ
        TABULATE    TXNX
        ASSIGN      1,0
        ASSIGN      2,0
        ASSIGN      3,0
        ASSIGN      4,0
        ASSIGN      5,0
        ASSIGN      6,0
        ENDMACRO

        SIMULATE
***********************************
*                                 *
*   CPU #1                        *
*                                 *
***********************************

 CPU1   GENERATE    ,,,X$MAXMP,,,F
*************************
* START FOR CPU1 TXNS   *
*************************
 STAR1  PRIORITY    9                   SET HIGH PRIORITY
        MARK        2                   ARRIVAL TIME OF TXN
        ASSIGN      1,1                 SET CPU ID = 1
```

```
       TRANSFER    .X$NREAD,WWW1,RRR1      READ OR WRITE TXN?
*************************
* READ TXN FROM CPU1    *
*************************
 RRR1  QUEUE       DIQ11                 READ TXN
       SEIZE       DRP11
       DEPART      DIQ11
       PRIORITY    0                    RESET PRIORITY
       ASSIGN      4+,X$REX             TIME FOR DIRECTORY SEARCH
       ASSIGN      7,X$REX
       ADVANCE     P7
       RELEASE     DRP11
       TRANSFER    .X$PIN1,NIN11,IND11 IS DATA IN L(1)?
*************************
* READ TXN FROM CPU1    *
* IS SATISFIED IN L(1)  *
*************************

 IND11 ASSIGN      11,0

*************************
* READ DATA FROM D11    *
*************************

 UQT   MACRO       DIQ11,DRP11,X$DEX11

*************************
* USE FINI MACRO        *
* THE TXN IS COMPLETED  *
*************************

FINI   MACRO
       TRANSFER    ,STAR1               THE TXN BECOMES A NEW TXN
*************************
* READ TXN FROM CPU1 IS *
* NOT SATISFIED IN L(1) *
*************************

 NIN11 QUEUE       DOQ11

*************************
* USE UTX TO USE        *
* THE LOCAL BUS LBUS1      *
*************************
 UTX   MACRO       LBUS1,DOQ11,X$BEXM
       TRANSFER    ,COMR                 GO TO COMMON CODE FOR READ
*************************
* WRITE TXN FROM CPU1   *
*************************
 WWW1  QUEUE       DIQ11
       TEST E      BV$BVA1,1            D11 OUT QUEUE AND DRP FREE?
       SAVEVALUE   DYL11+,1             SAVE SPACE IN OUT Q
       SEIZE       DRP11
       PRIORITY    0                     RESET PRIORITY
       DEPART      DIQ11
```

```
      ASSIGN     4+,X$DEX11          TIME FOR WRITING DATA
      ASSIGN     7,X$DEX11
      ADVANCE    P7
      RELEASE    DRP11
      SPLIT      1,STB1               CREATE A STORE-BEHIND TXN
************************
* WRITE TXN IS COMPLETED*
************************
 FINI  MACRO
      TRANSFER   ,STAR1              BECOMES A NEW TXN FROM CPU1
************************
* STORE-BEHIND TXN     *
************************
 STB1  QUEUE     DYQ11                PUT TXN IN DATA QUEUE

      TEST E     BV$BVA2,1          K1 IN-Q AND LBUS1 FREE ?
      SAVEVALUE  KXL1+,1            RESERVE SPACE IN IN-Q

************************
* USE LBUS1 TO SEND TXN   *
* FROM D11 TO K1        *
************************
 UTX   MACRO     LBUS1,DYQ11,X$BEXD1
      SAVEVALUE  DYL11-,1            RELEASE SPACE IN D11
      TRANSFER   ,COMW               TO COMMON CODE FOR WRITE

************************
* COMMON CODE FOR      *
* READ TO LOWER LEVELS  *
* JOINED BY ALL CPUS    *
************************

 COMR  ASSIGN     11,0              DUMMY STATEMENT
************************
* USE K1               *
************************
 UQTQ  MACRO      KIQ1,KRP1,KOQ1,X$KEX
************************
* USE GLOBAL BUS GBUS   *
************************
 UTX   MACRO      GBUS,KOQ1,X$BEXM
************************
* USE K2               *
************************
 UQTQ  MACRO      KIQ2,KRP2,KOQ2,X$KEX
************************
*  USE LOCAL BUS LBUS2    *
************************
 UTX   MACRO      LBUS2,KOQ2,X$BEXM
************************
* USE R2 TO SEE IF DATA *
* IS IN L(2)           *
************************

 UQT   MACRO      RIQ2,RRP2,X$REX
```

```
       TRANSFER   .X$PIN2,NIN2,INL2      IS DATA IN L(2)?
*************************
* DATA IS NOT FOUND IN  *
* L(2)                  *
*************************

 NIN2  QUEUE      ROQ2

*************************
* USE LBUS2 SEND TXN TO   *
* K2                    *
*************************
 UTX   MACRO      LBUS2,ROQ2,X$BEXM
*************************
*  SERVICED BY K2       *
*************************
 UQTQ  MACRO      KIQ2,KRP2,KOQ2,X$KEX
*************************
* USE GBUS SEND TXN  TO   *
* K3                    *
*************************
 UTX   MACRO      GBUS,KOQ2,X$BEXM
*************************
* SERVICED BY K3        *
*************************
 UQTQ  MACRO      KIQ3,KRP3,KOQ3,X$KEX
*************************
* USE LBUS3 SEND TXN TO   *
* R3                    *
*************************
 UTX   MACRO      LBUS3,KOQ3,X$BEXM
*************************
* SEARCH DIRECTORY IN   *
* R3 FOR DATA           *
*************************

 UQT   MACRO      RIQ3,RRP3,X$REX
       TRANSFER   ,INL3                DATA IS IN L(3)
***********************************
* DATA IS FOUND IN L(2), READ THE *
* DATA AND SEND IT UP TO L(1)     *
***********************************

 INL2  QUEUE      ROQ2

*************************
* SEND TXN TO DEVICE    *
* VIA LBUS2               *
*************************
 UTX   MACRO      LBUS2,ROQ2,X$BEXM

**********************************
* IS DATA IN D11 OR D12?         *
**********************************
```

```
       TRANSFER   .5,RRR21,RRR22

*********************
* DATA IS IN D11    *
*********************

 RRR21 QUEUE      DIQ21              QUEUE TO RETRIEVE DATA
       TEST E     BV$BVA3,1         D21 OUT-Q AND DRP21 FREE?
       SAVEVALUE  DYL21+,1          SAVE SPACE IN D21 OUT-Q

*************************
* USE D21 TO RETRIEVE   *
* THE DATA              *
*************************
 UTX   MACRO      DRP21,DIQ21,X$DEX21  RETRIEVE THE DATA
       QUEUE      DYQ21              PUT DATA IN SLOT

       TEST E     BV$BVA4,1         K2 IN-Q AND LBUS2 FREE?
       SAVEVALUE  KXL2+,1           RESERVE K2 IN-Q SLOT

*************************
* USE LBUS2 SEND DATA TO  *
* K2                    *
*************************
 UTX   MACRO      LBUS2,DYQ21,X$BEXD1

       SAVEVALUE  DYL21-,1          RELEASE SLOT IN D21 OUT-QUEUE
       TRANSFER   ,RTF2             TO CODE FOR READ-THROUGH FROM L(2)

********************
* DATA IS IN D22   *
********************

 RRR22 QUEUE      DIQ22
       TEST E     BV$BVA21,1
       SAVEVALUE  DYL22+,1

 UTX   MACRO      DRP22,DIQ22,X$DEX22

       QUEUE      DYQ22
       TEST E     BV$BVA4,1
       SAVEVALUE  KXL2+,1

 UTX   MACRO      LBUS2,DYQ22,X$BEXD1

       SAVEVALUE  DYL22-,1
       TRANSFER   ,RTF2

***********************************
* READ THROUGH FROM LEVEL L(2)    *
***********************************

 RTF2  ASSIGN     11,0
```

```
*******************************
* SERVICED BY K2              *
*******************************
 UQDQ  MACRO      KXQ2,KXL2-,KYQ2,KYL2+,KRP2,X$KEX,BV$BVA5

       TEST E     BV$BVA6,1         K1 IN-Q AND GBUS FREE?
       SAVEVALUE  KXL1+,1           RESERVE K1 IN-Q SLOT

*************************
* USE GBUS TO SEND DATA TO*
* K1                    *
*************************
 UTX   MACRO      GBUS,KYQ2,X$BEXD1
       SAVEVALUE  KYL2-,1           RELEASE SLOT IN K2

***********************************
* STORE DATA INTO L(1) AS A RESULT*
* OF READ-THROUGH                 *
***********************************

 STOR1 ASSIGN     11,0

*************************
*  SERVICED BY K1       *
*************************

 UQD   MACRO      KXQ1,KXL1-,,KYL1+,KRP1,X$KEX,BV$BVA20


*********************************
* SEND TO D11 OR D12            *
*********************************

       SPLIT      1,FN$WICHW,1      WHICH DATA CACHE TO GO?
       TERMINATE

********************
* STORE TO D11     *
********************

 WWW11 ASSIGN     11,0              WRITE TO D11
       QUEUE      KYQ1
       TEST E     BV$BVA7,1         SPACE IN D11 IN-Q AND LBUS1 FREE?
       SAVEVALUE  DXL11+,1          YES, RESERVE A SLOT

*************************
* SEND TXN TO D11 VIA   *
* LBUS1                   *
*************************

 UTX   MACRO      LBUS1,KYQ1,X$BEXD1

       SAVEVALUE  KYL1-,1           RELEASE K1 SLOT

*************************
```

```
* WRITE DATA TO D11     *
*************************

 UQT   MACRO       DXQ11,DRP11,X$DEX11

       SAVEVALUE   DXL11-,1
       TRANSFER    .X$POV1,NOV11,OVL11      ANY OVERFLOW FROM L(1)?

*************************
* NO OVERFLOW FROM L(1) *
*************************

 NOV11 ASSIGN      11,0

*************************
* THE READ TXN HAS ENDED*
*************************

 FINI  MACRO
       TRANSFER    ,STAR1

*************************
* THERE IS OVERFLOW FRCM*
* L(1), END THE READ    *
* TXN, AT THE SAME TIME *
* HANDLE THE OVERFLOW   *
*************************

 OVL11 SPLIT       1,OVF11             GOT OVERFLOW HANDLING
 FINI  MACRO                          AT THE SAME TIME END THE TXN
       TRANSFER    ,STAR1

*************************
* OVERFLOW HANDLING FOR *
* D11                   *
*************************

 OVF11 ASSIGN      11,0

 UQT   MACRO       DOQ11,LBUS1,X$BEXM
       TRANSFER    ,OVL1              GOTO COMMON CODE FOR OVERFLOW

*************************************
*  WWW12                            *
*************************************

*************************************
*   WWW13                            *
*************************************

 WWW12 ASSIGN      11,0
 WWW13 ASSIGN      11,0
***********************************
* COMMON CODE FOR OVERFLOW FROM   *
* L(1)                            *
```

```
***********************************

OVL1   ASSIGN     11,C

****************************************
* USE  K1, THEN GBUS, THEN K2   *
* THEN LBUS2, THEN USE R2                *
****************************************

 UQTQ  MACRO      KIQ1,KRP1,KOQ1,X$KEX

 UTX   MACRO      GBUS,KOQ1,X$BEXM

 UQTQ  MACRO      KIQ2,KRP2,KOQ2,X$KEX

 UTX   MACRO      LBUS2,KOQ2,X$BEXM

       QUEUE      RIQ2

 UTX   MACRO      RRP2,RIQ2,X$REX

       TERMINATE

*************************
* DATA IS FOUND IN L(3) *
*************************

 INL3  QUEUE      ROQ3

*************************
* USE LBUS3 SEND TXN TO   *
* D31                   *
*************************

 UTX   MACRO      LBUS3,ROQ3,X$BEXM

*****************************
* READ FROM D31 OR D32?     *
*****************************

       TRANSFER   .5,RRR31,RRR32

********************
* READ FROM D31    *
********************

 RRR31 QUEUE      DIQ31
       TEST E     BV$BVA8,1          SPACE IN D31 OUT-Q AND DRP31 FREE?
       SAVEVALUE  DYL31+,1

*************************
* READ DATA FROM D31    *
*************************
```

```
 UTX   MACRO      DRP31,DIQ31,X$DEX31

       QUEUE      DYQ31
       TEST E     BV$BVA9,1          SPACE IN K3 IN-Q AND LBUS3 FREE?
       SAVEVALUE  KXL3+,1            YES, RESERVE SLOT

*************************
* USE LBUS3 SEND DATA TO    *
* K3                    *
*************************

 UTX   MACRO      LBUS3,DYQ31,X$BEXD2

       SAVEVALUE  DYL31-,1
       TRANSFER   ,RTF3              GO TO READ-THROUGH FROM L(3)

********************
* READ FROM D32    *
********************

 RRB32 QUEUE      DIQ32
       TEST E     BV$BVA22,1
       SAVEVALUE  DYL32+,1

 UTX   MACRO      DRP32,DIQ32,X$DEX32

       QUEUE      DYQ32
       TEST E     BV$BVA9,1
       SAVEVALUE  KXL3+,1

 UTX   MACRO      LBUS3,DYQ32,X$BEXD2

       SAVEVALUE  DYL32-,1
       TRANSFER   ,RTF3

***********************************
* READ-THROUGH FROM L(3) DATA IS  *
* SENT TO L(2) AND L(1) AT THE    *
* SAME TIME                       *
***********************************

 RTF3  ASSIGN     11,0

*************************
*  SERVICED BY K3       *
*************************

 UQDQ  MACRO      KXQ3,KXL3-,KYQ3,KYL3+,KRP3,X$KEX,BV$BVA10

       TEST E     BV$RTOK,1          L(1) & L(2) READY & GBUS FREE?
       SAVEVALUE  KXL1+,1
       SAVEVALUE  KXL2+,1

*************************
* BOTH L(1) AND L(2)    *
* READY TO ACCEPT DATA  *
* FROM GBUS             *
*************************

 UTX    MACRO       GBUS,KYQ3,X$BEXD2

        SAVEVALUE   KYL3-,1
        SPLIT       1,STOR1             READ-THROUGH TO L(1)

*************************
* READ-THROUGH TO L(2)  *
*************************

 STOR2 ASSIGN       11,0

*************************
* SERVICED BY K2        *
*************************

 UQDQ   MACRO       KXQ2,KXL2-,KYQ2,KYL2+,KRP2,X$KEX,BV$BVA5

        TEST E      BV$BVA11,1          SPACE IN R2 IN-Q AND LBUS2 FREE?
        SAVEVALUE   RXL2+,1             YES, RESERVE SLOT

*************************
* USE LBUS2 SEND TO R2  *
*************************

 UTX    MACRO       LBUS2,KYQ2,X$BEXD2

        SAVEVALUE   KYL2-,1             FREE SLOT IN K2

*************************
* SERVICED BY R2        *
*************************

 UQD    MACRO       RXQ2,RXL2-,,RYL2+,RRP2,X$REX,BV$BVA12

        SPLIT       1,OVH2              HANDLE ANY OVERFLOW

********************************
* STORE INTO D21 OR D22?       *
********************************

        TRANSFER    .5,SSS21,SSS22

*************************
* STORE INTO D21        *
*************************

 SSS21 QUEUE        RYQ2
        TEST E      BV$BVA13,1          D21 IN-Q AND LBUS2 FREE?
        SAVEVALUE   DXL21+,1            YES, RESERVE THE SPACE

*******************************
```

```
* SEND DATA TO D21 VIA BUS    *
*******************************

 UTX    MACRO        LBUS2,RYQ2,X$BEXD2

        SAVEVALUE    RYL2-,1          RELEASE SPACE IN R2

 UQT    MACRO        DXQ21,DRP21,X$DEX21

        SAVEVALUE    DXL21-,1
        TERMINATE

**************************
* STORE INTO D22         *
**************************

 SSS22 QUEUE         RYQ2
        TEST E       BV$BVA23,1
        SAVEVALUE    DXL22+,1

 UTX    MACRO        LBUS2,RYQ2,X$BEXD2

        SAVEVALUE    RYL2-,1

 UQT    MACRO        DXQ22,DRP22,X$DEX22

        SAVEVALUE    DXL22-,1
        TERMINATE

*******************************
* HAND. ANY OVERF. FROM L(2) *
*******************************

 OVH2   TRANSFER     .X$POV2,NOV2,OVL2
 OVL2   QUEUE        ROQ2

**************************
* USE LBUS2, USE K2, USE  *
* GBUS, USE K3, USE LBUS3,  *
* THEN USE R3            *
**************************

 UTX    MACRO        LBUS2,ROQ2,X$BEXM

 UQTQ   MACRO        KIQ2,KRP2,KOQ2,X$KEX

 UTX    MACRO        GBUS,KOQ2,X$BEXM

 UQTQ   MACRO        KIQ3,KRP3,KOQ3,X$KEX

 UTX    MACRO        LBUS3,KOQ3,X$BEXM

 UQT    MACRO        RIQ3,RRP3,X$REX

 NOV2   TERMINATE
```

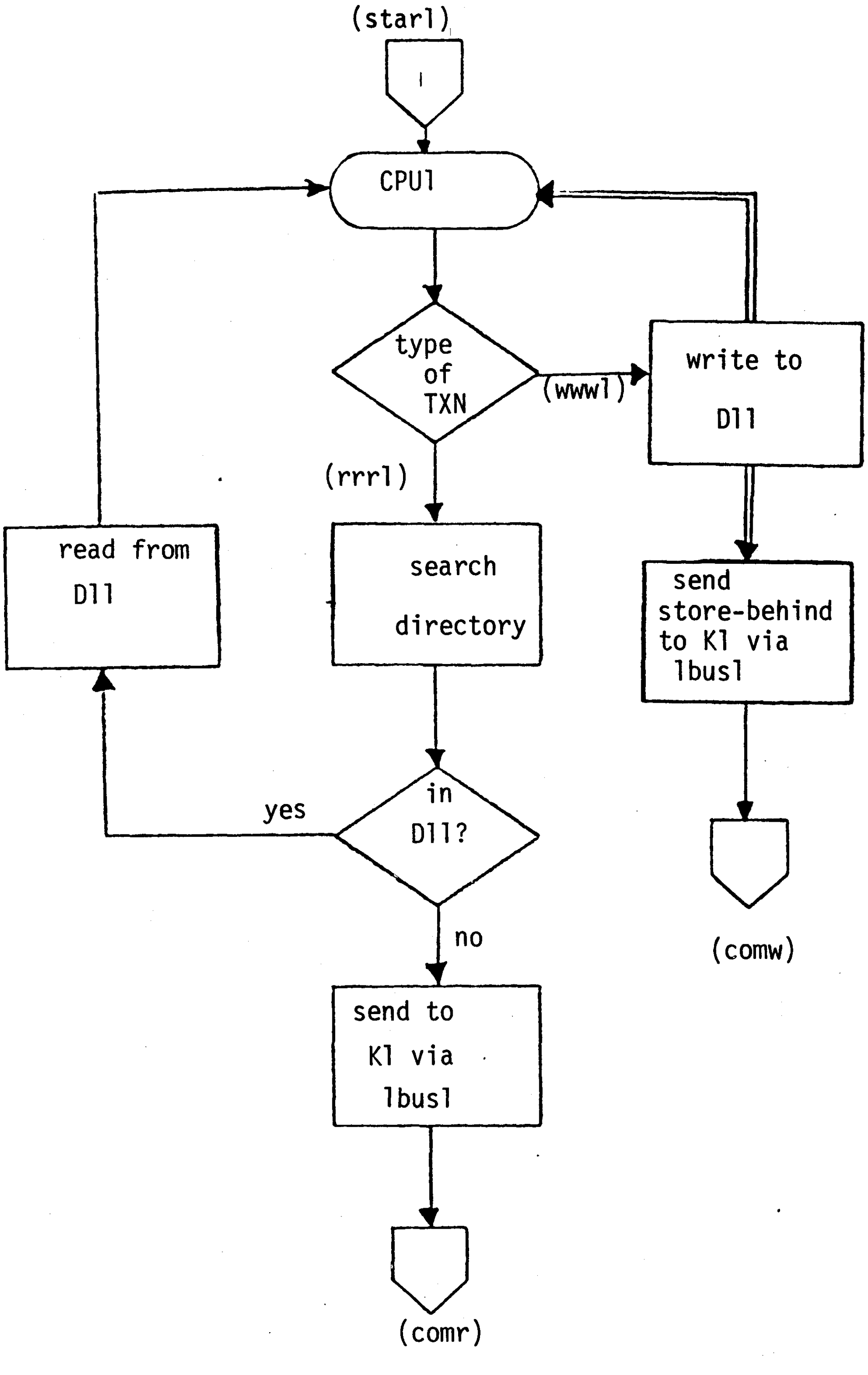
```
*************************
* COMMCN CODE FOR WRITE *
* TO LOWER LEVELS       *
*************************
 COMW  ASSIGN     11,0                 DUMMY STATEMENT

*************************
* SERVICED BY K1        *
*************************

 UQDQ  MACRO      KXQ1,KXL1-,KYQ1,KYL1+,KRP1,X$KEX,BV$BVA14

       TEST E     BV$BVA15,1           K2 IN-Q AND GBUS FREE?
       SAVEVALUE  KXL2+,1

*************************
* USE GBUS               *
*************************

 UTX   MACRO      GBUS,KYQ1,X$BEXD1
       SAVEVALUE  KYL1-,1

*************************
* SERVICED BY K2        *
*************************

 UQDQ  MACRO      KXQ2,KXL2-,KYQ2,KYL2+,KRP2,X$KEX,BV$BVA5

       TEST E     BV$BVA11,1           R2 IN-Q AND LBUS2 FREE?
       SAVEVALUE  RXL2+,1

*************************
* USE LBUS2              *
*************************

 UTX   MACRO      LBUS2,KYQ2,X$BEXD1
       SAVEVALUE  KYL2-,1

*************************
* SERVICED BY R2        *
*************************

 UQD   MACRO      RXQ2,RXL2-,,RYL2+,RRP2,X$REX,BV$BVA12

*********************************
* SERVED BY D21 OR D22?         *
*********************************

       TRANSFER   .5,SWS21,SWS22


*************************
* SERVICED BY D21       *
*************************
```

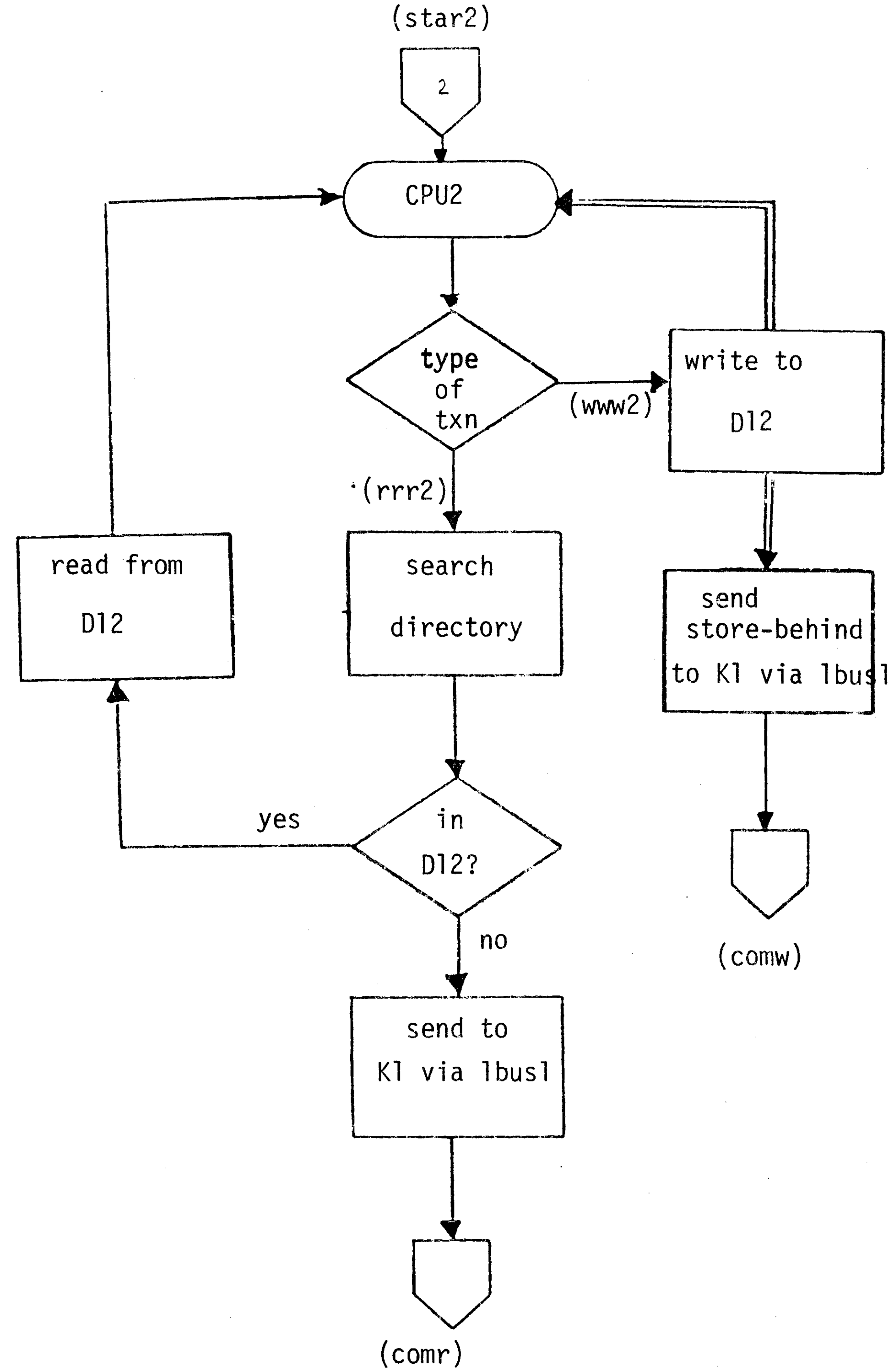
```
 SWS21 QUEUE       RYQ2
       TEST E      BV$BVA13,1
       SAVEVALUE   DXL21+,1

 UTX   MACRO       LBUS2,RYQ2,X$BEXD1

       SAVEVALUE   RYL2-,1
 UQDQ  MACRO       DXQ21,DXL21-,DYQ21,DYL21+,DRP21,X$DEX21,BV$BVA3

       TEST E      BV$BVA4,1          K2 IN-Q AND LBUS2 FREE?
       SAVEVALUE   KXL2+,1

*************************
* USE LBUS2 SEND TO K2  *
*************************

 UTX   MACRO       LBUS2,DYQ21,X$BEXD2

       SAVEVALUE   DYL21-,1
       SPLIT       1,ACK2             PREPARE TO SEND ACK TO L(1)

       TRANSFER    ,STB23             GO TO STORE-BEHIND TO L(3)

******************************
* SEND ACK TO L(1)           *
******************************

 ACK2  QUEUE       DOQ21

 UTX   MACRO       LBUS2,DOQ21,X$BEXM

       TRANSFER    ,ACK21

*****************************
* SERVICED BY D22           *
*****************************

 SWS22 QUEUE       RYQ2
       TEST E      BV$BVA23,1
       SAVEVALUE   DXL22+,1

 UTX   MACRO       LBUS2,RYQ2,X$BEXD1

       SAVEVALUE   RYL2-,1

 UQDQ  MACRO       DXQ22,DXL22-,DYQ22,DYL22+,DRP22,X$DEX22,BV$BVA21

       TEST E      BV$BVA4,1
       SAVEVALUE   KXL2+,1

 UTX   MACRO       LBUS2,DYQ22,X$BEXD2

       SAVEVALUE   DYL22-,1
       SPLIT       1,ACK3
```

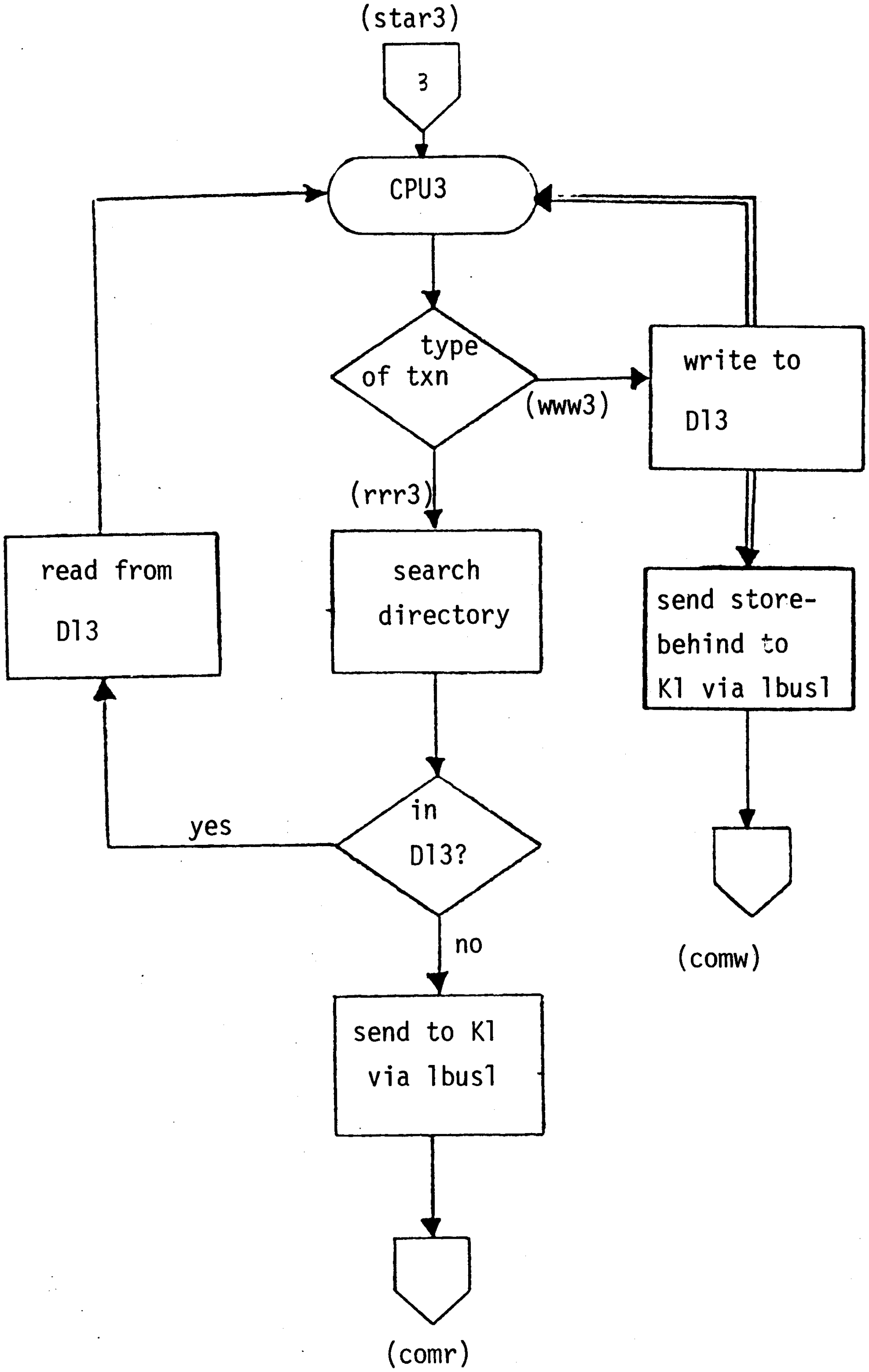
```
      TRANSFER   ,STB23

 ACK3 QUEUE      DOQ22

 UTX  MACRO      LBUS2,DOQ22,X$BEXM

      TRANSFER   ,ACK21

*************************
* STORE-BEHIND FROM     *
* L(2) TO L(3)          *
*************************

 STB23 ASSIGN    11,0

 UQDQ  MACRO     KXQ2,KXL2-,KYQ2,KYL2+,KRP2,X$KEX,BV$BVA5

       TEST E    BV$BVA16,1       K3 IN-Q AND GBUS FREE?
       SAVEVALUE KXL3+,1

 UTX   MACRO     GBUS,KYQ2,X$BEXD2

       SAVEVALUE KYL2-,1

 UQDQ  MACRO     KXQ3,KXL3-,KYQ3,KYL3+,KRP3,X$KEX,BV$BVA10

       TEST E    BV$BVA17,1       R3 IN-Q AND LBUS3 FREE?
       SAVEVALUE RXL3+,1

 UTX   MACRO     LBUS3,KYQ3,X$BEXD2

       SAVEVALUE KYL3-,1

 UQD   MACRO     RXQ3,RXL3-,,RYL3+,RRP3,X$REX,BV$BVA18

**********************************
* SERVICED BY D31 OR D32?        *
**********************************

       TRANSFER  .5,SWS31,SWS32

********************
* SERV. BY D31     *
********************

 SWS31 QUEUE     RYQ3
       TEST E    BV$BVA19,1
       SAVEVALUE DXL31+,1

 UTX   MACRO     LBUS3,RYQ3,X$BEXD2

       SAVEVALUE RYL3-,1
 UQT   MACRO        DXQ31,DRP31,X$DEX31

       SAVEVALUE    DXL31-,1

 UQT   MACRO        DOQ31,LBUS3,X$BEXM
       TRANSFER     ,ACK22

********************
* SERV. BY D32  *
********************

 SWS32 QUEUE        RYQ3
       TEST E       BV$BVA24,1
       SAVEVALUE    DXL32+,1

 UTX   MACRO        LBUS3,RYQ3,X$BEXD2

       SAVEVALUE    RYL3-,1

 UQT   MACRO        DXQ32,DRP32,X$DEX32

       SAVEVALUE    DXL32-,1

 UQT   MACRO        DOQ32,LBUS3,X$BEXM

       TRANSFER     ,ACK22

************************************
* ACK FROM L(2) TO L(3)            *
************************************

 ACK22 ASSIGN       11,0

 UQTQ  MACRO        KIQ3,KRP3,KOQ3,X$KEX

 UTX   MACRO        GBUS,KOQ3,X$BEXM

 UQTQ  MACRO        KIQ2,KRP2,KOQ2,X$KEX

 UTX   MACRO        LBUS2,KOQ2,X$BEXM

 UQTQ  MACRO        RIQ2,RRP2,ROQ2,X$REX

 UTX   MACRO        LBUS2,ROQ2,X$BEXM
       TRANSFER     ,ACK21

************************************
* ACK FROM L(2) TO L(1)            *
************************************

 ACK21 ASSIGN       11,0

 UQTQ  MACRO        KIQ2,KRP2,KOQ2,X$KEX
 UTX    MACRO      GDUS,KOQ2,X$BEXM

 UQTQ   MACRO      KIQ1,KRP1,KOQ1,X$KEX

 UTX    MACRO      LBUS1,KOQ1,X$DEXM
        SPLIT      1,FN$WICHA,1
        TERMINATE
 AAA11 ASSIGN      11,0
 AAA12 ASSIGN      11,0
 AAA13 ASSIGN      11,0
        QUEUE      DIQ11
        SEIZE      DRP11
        DEPART     DIQ11
        ASSIGN     4+,X$REX
        ASSIGN     7,X$REX
        ADVANCE    P7
        RELEASE    DRP11
        TERMINATE

******************
* AAA12          *
******************

******************
*  AAA13         *
******************


**********************************
*                                *
*  TIMER SEGMENT - TIME UNIT  IS *
*  ONE NANOSECOND                *
**********************************

        GENERATE   X$TIMER
        TERMINATE  1

        START      1
        END
/*
```

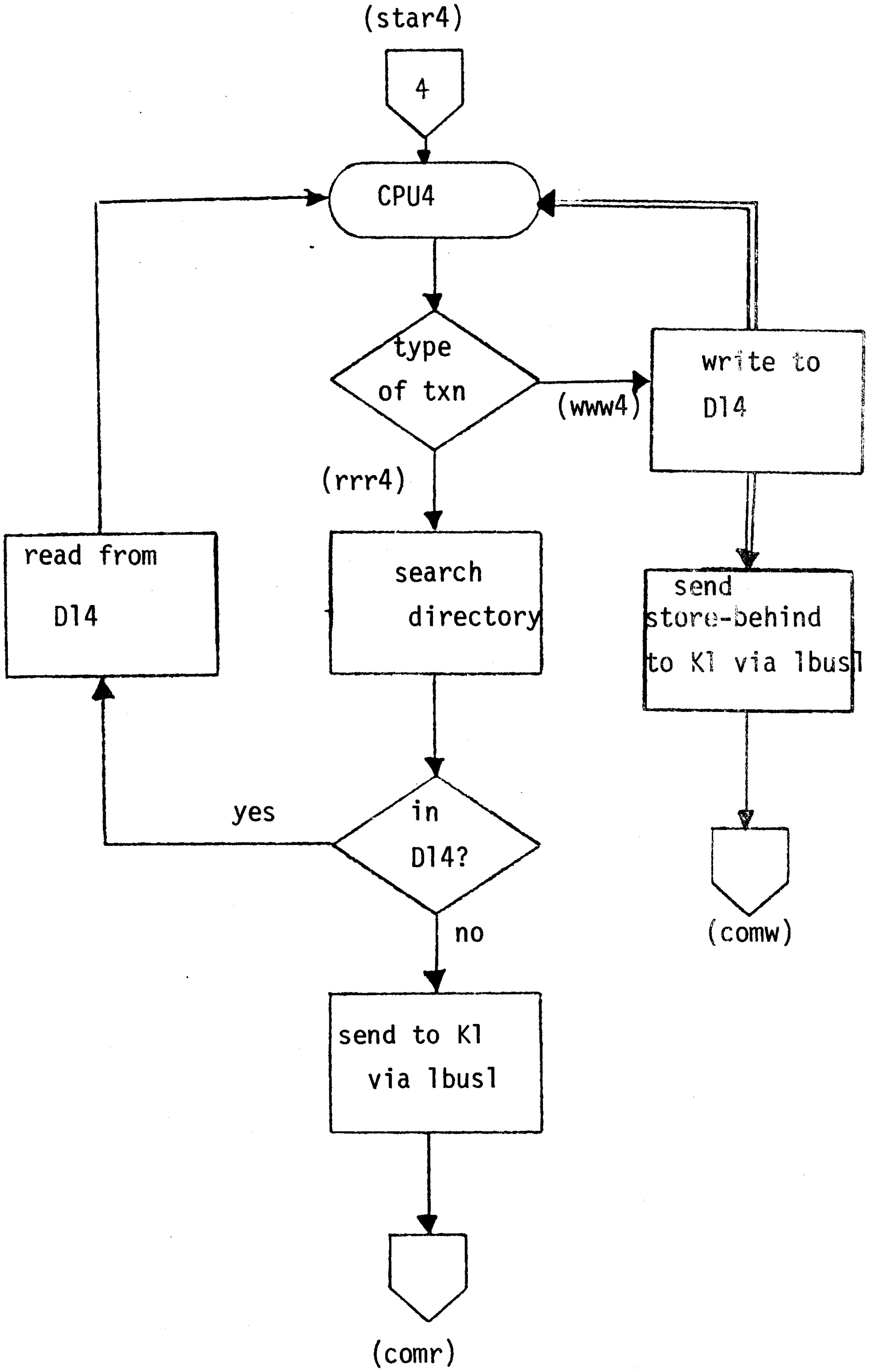
Appendix B

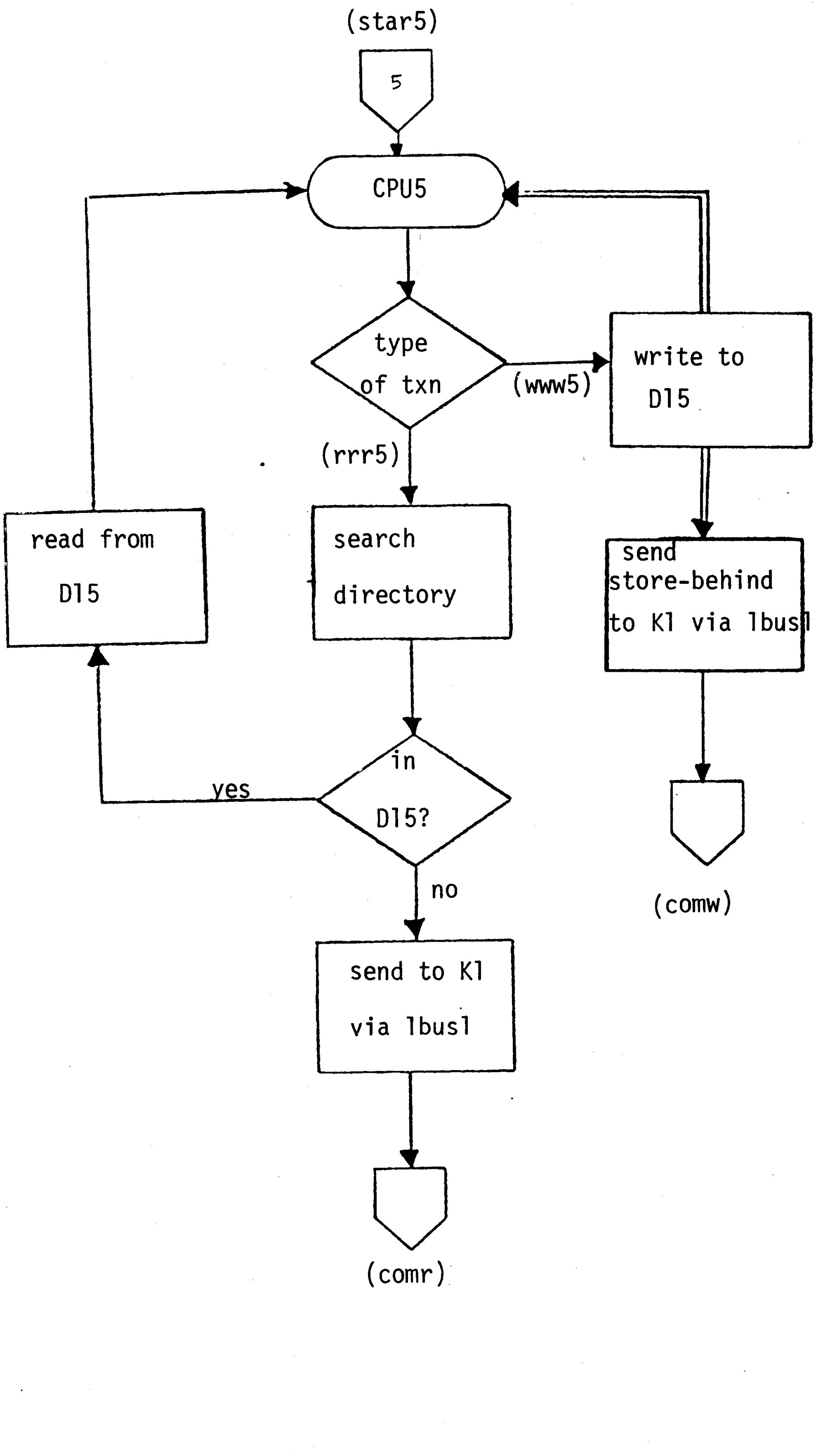
FLOW CHART OF THE P5L4 MODEL

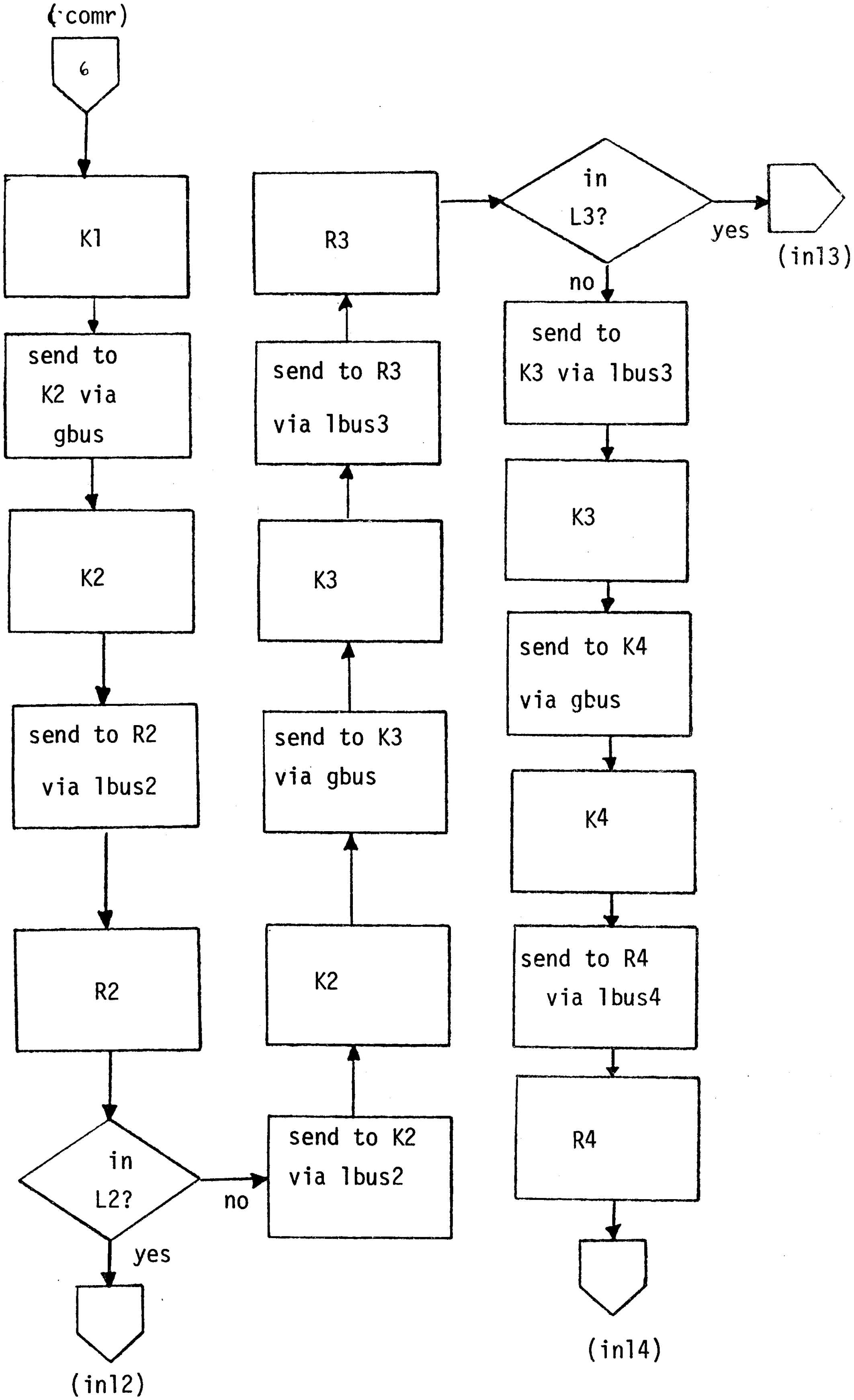

(starl)
1
CPU1
type of TXN
(wwwl)
write to D11
send store-behind to K1 via lbus1
(comw)
(rrrl)
search directory
in D11?
yes
read from D11
no
send to K1 via lbus1
(comr)

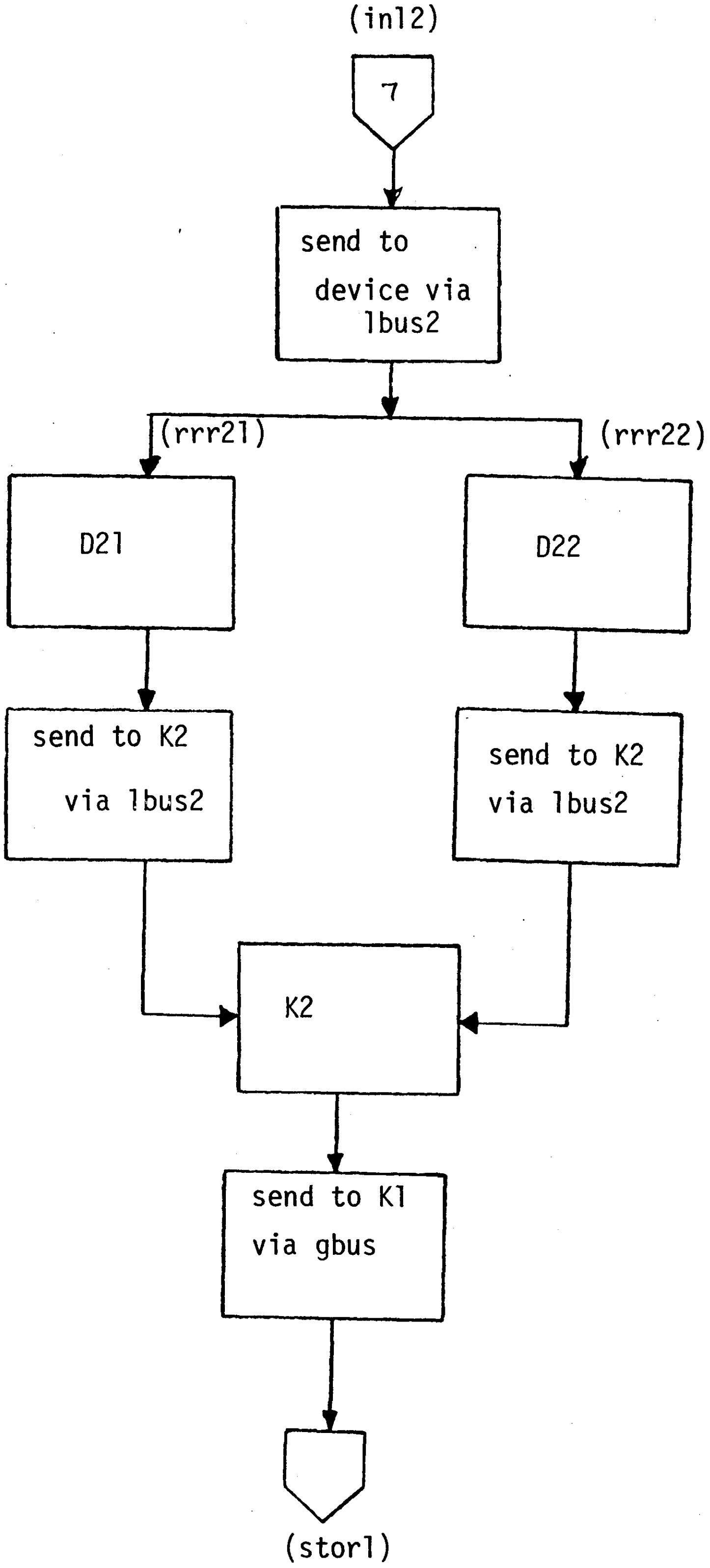

(star2)
2
CPU2
type of txn
(www2)
write to D12
(rrr2)
read from D12
search directory
send store-behind to K1 via lbus1
yes
in D12?
no
(comw)
send to K1 via lbus1
(comr)

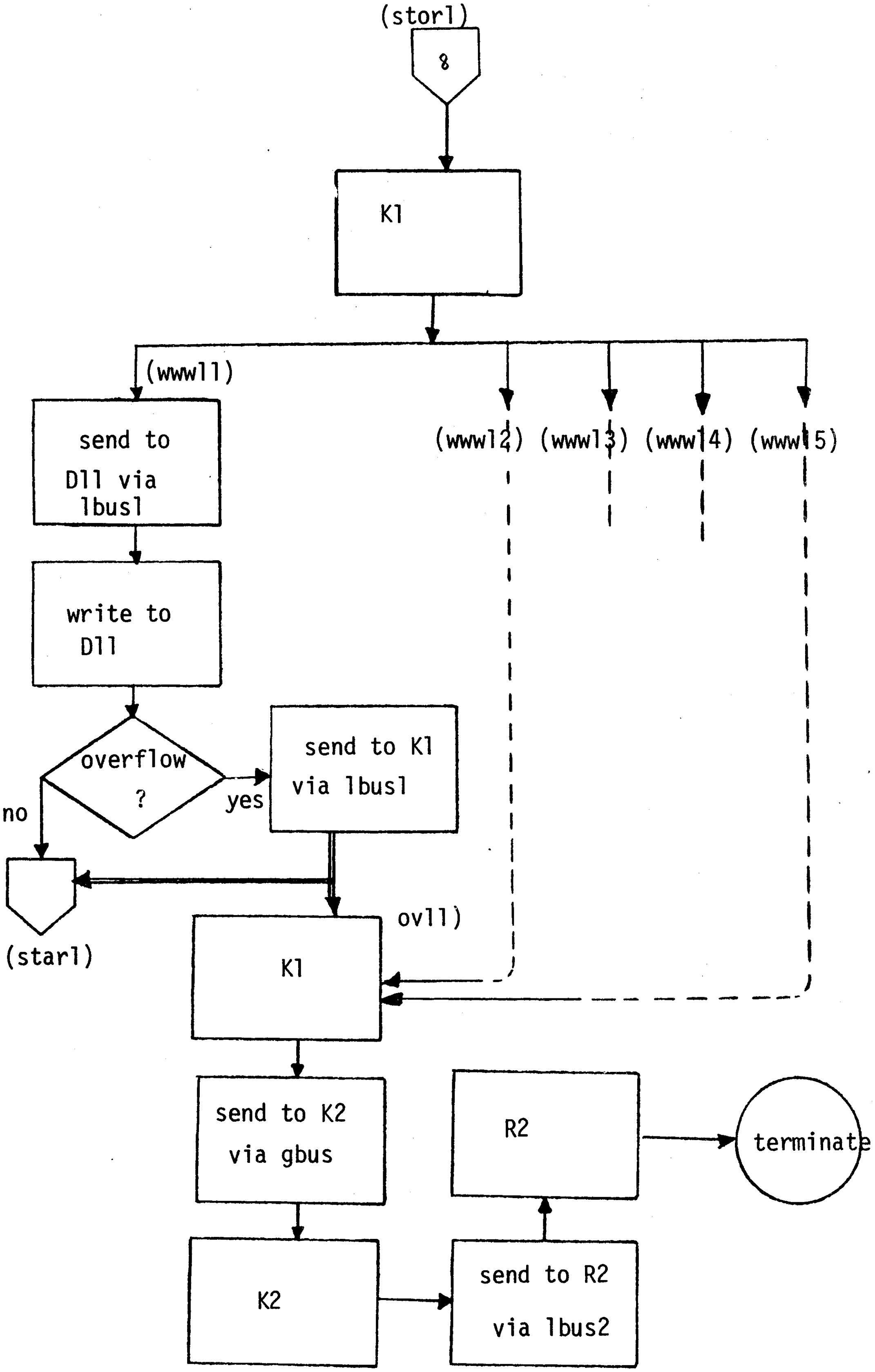

(star3)
3
CPU3
type of txn
(www3)
write to D13
(rrr3)
search directory
read from D13
send store-behind to K1 via 1bus1
in D13?
yes
no
send to K1 via 1bus1
(comw)
(comr)

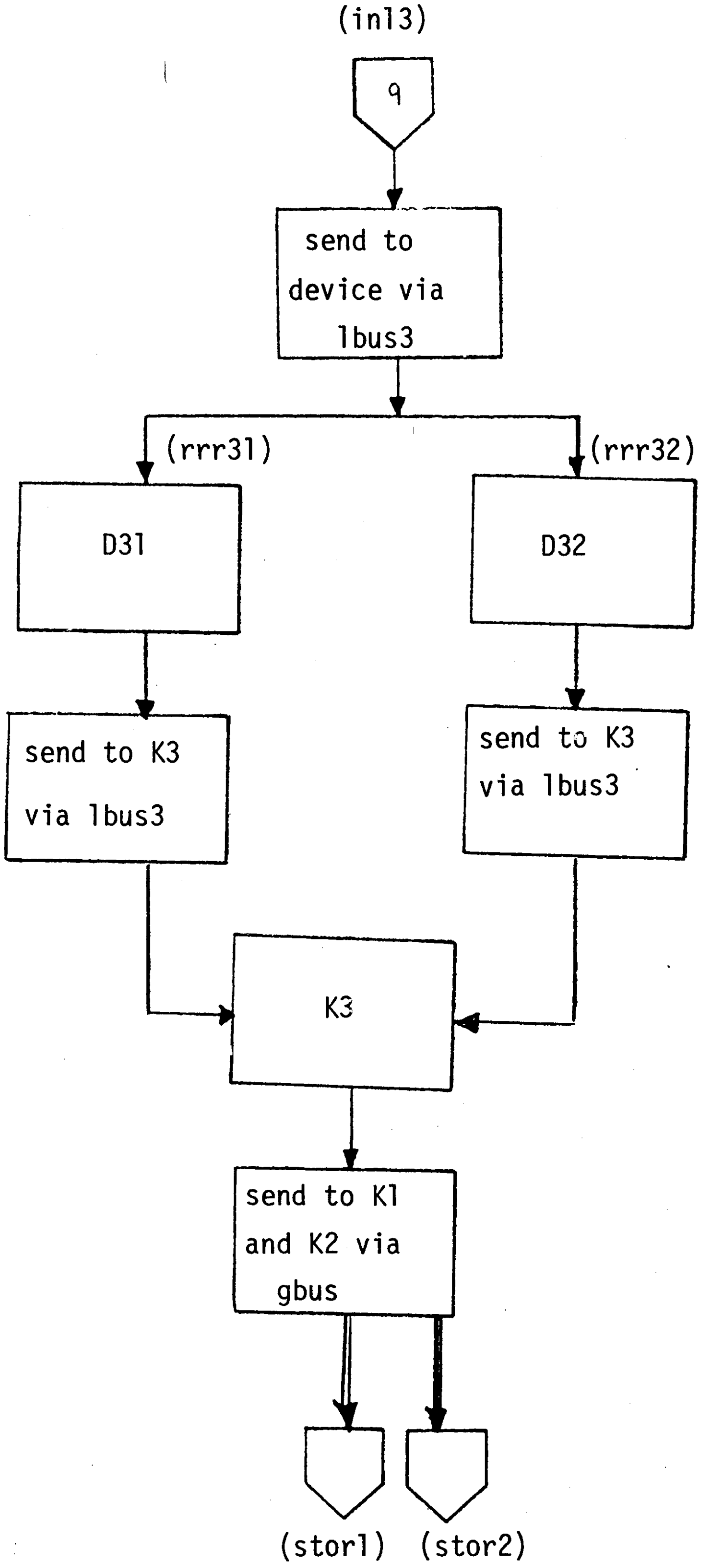

(star4)
4
CPU4
type of txn
(www4)
write to D14
(rrr4)
read from D14
search directory
send store-behind to K1 via lbus1
yes
in D14?
no
(comw)
send to K1 via lbus1
(comr)

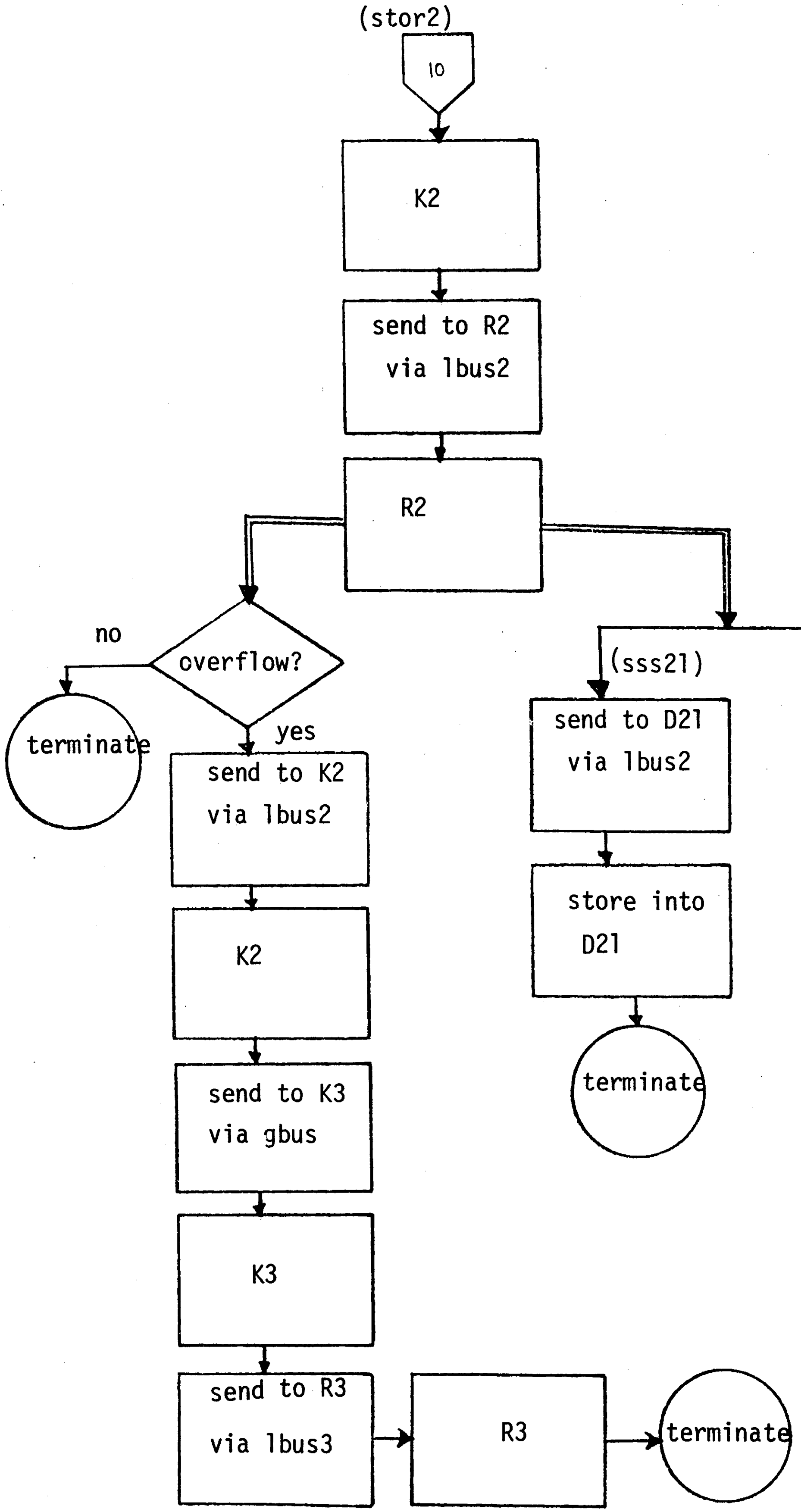

(star5)
5
CPU5
type of txn
(www5)
write to D15
(rrr5)
read from D15
search directory
send store-behind to K1 via lbus1
in D15?
yes
no
(comw)
send to K1 via lbus1
(comr)

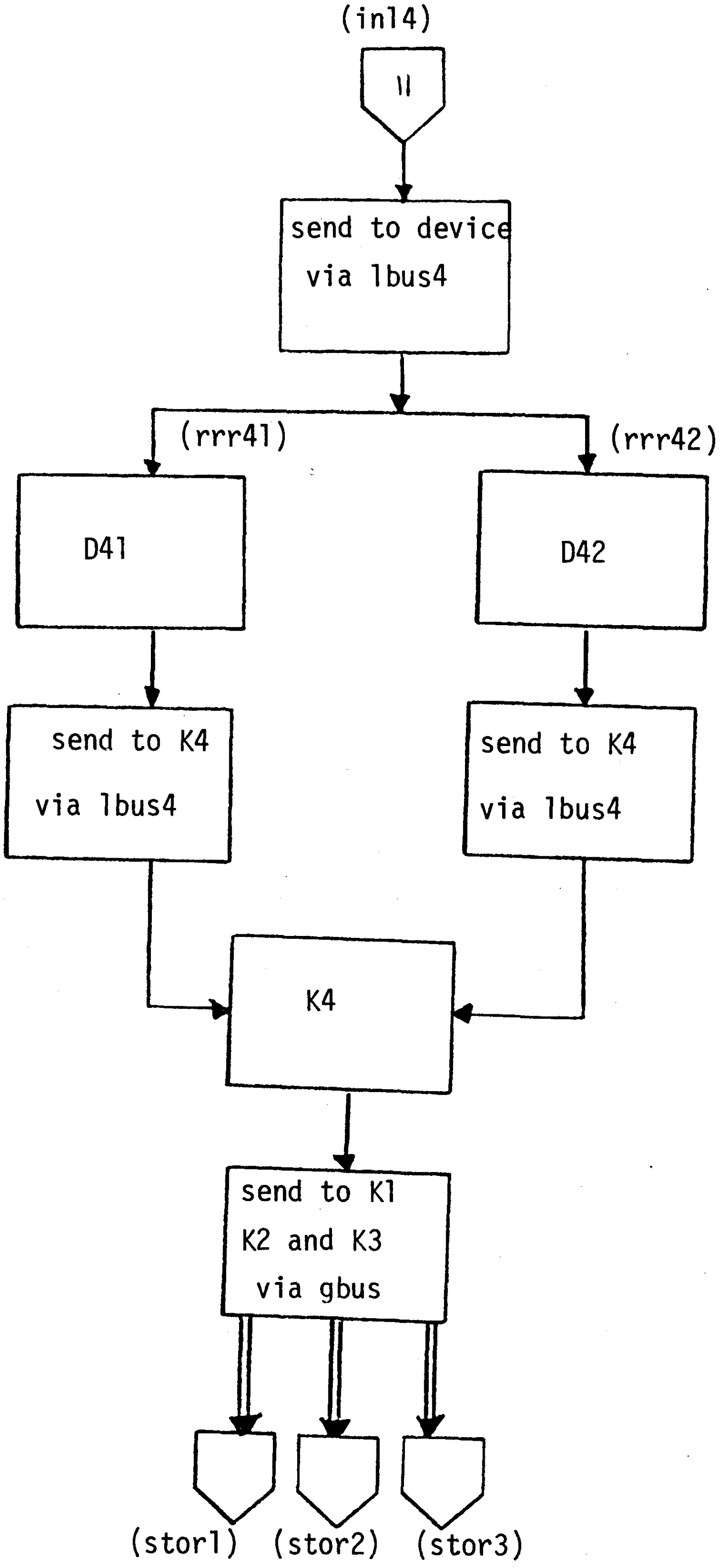

(comr)
6
K1
send to K2 via gbus
K2
send to R2 via lbus2
R2
in L2?
yes
no
(inl2)
send to K2 via lbus2
K2
send to K3 via gbus
K3
send to R3 via lbus3
R3
in L3?
yes
(inl3)
no
send to K3 via lbus3
K3
send to K4 via gbus
K4
send to R4 via lbus4
R4
(inl4)

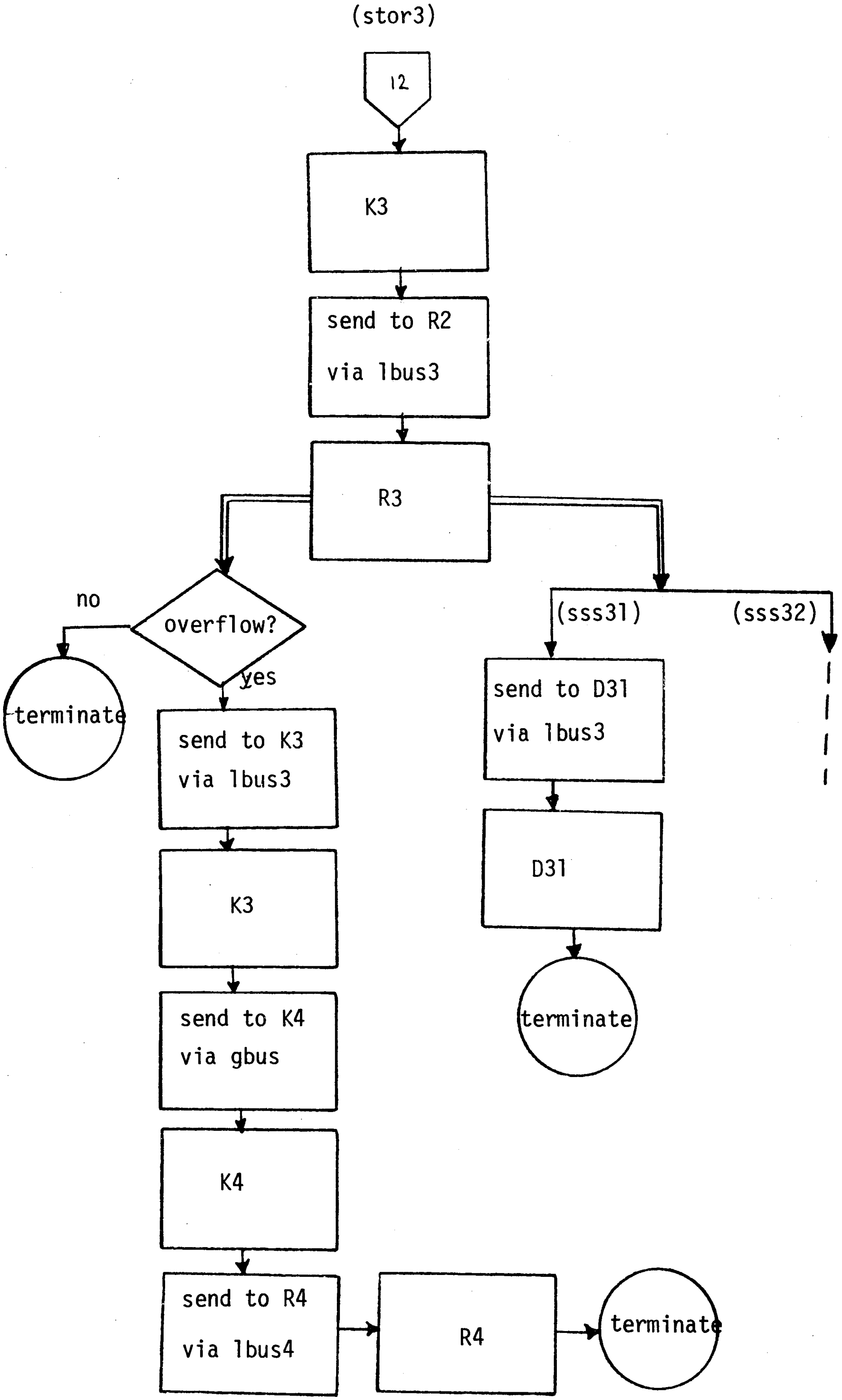

(inl2)
7
send to device via lbus2
(rrr21)
(rrr22)
D21
D22
send to K2 via lbus2
send to K2 via lbus2
K2
send to K1 via gbus
(storl)

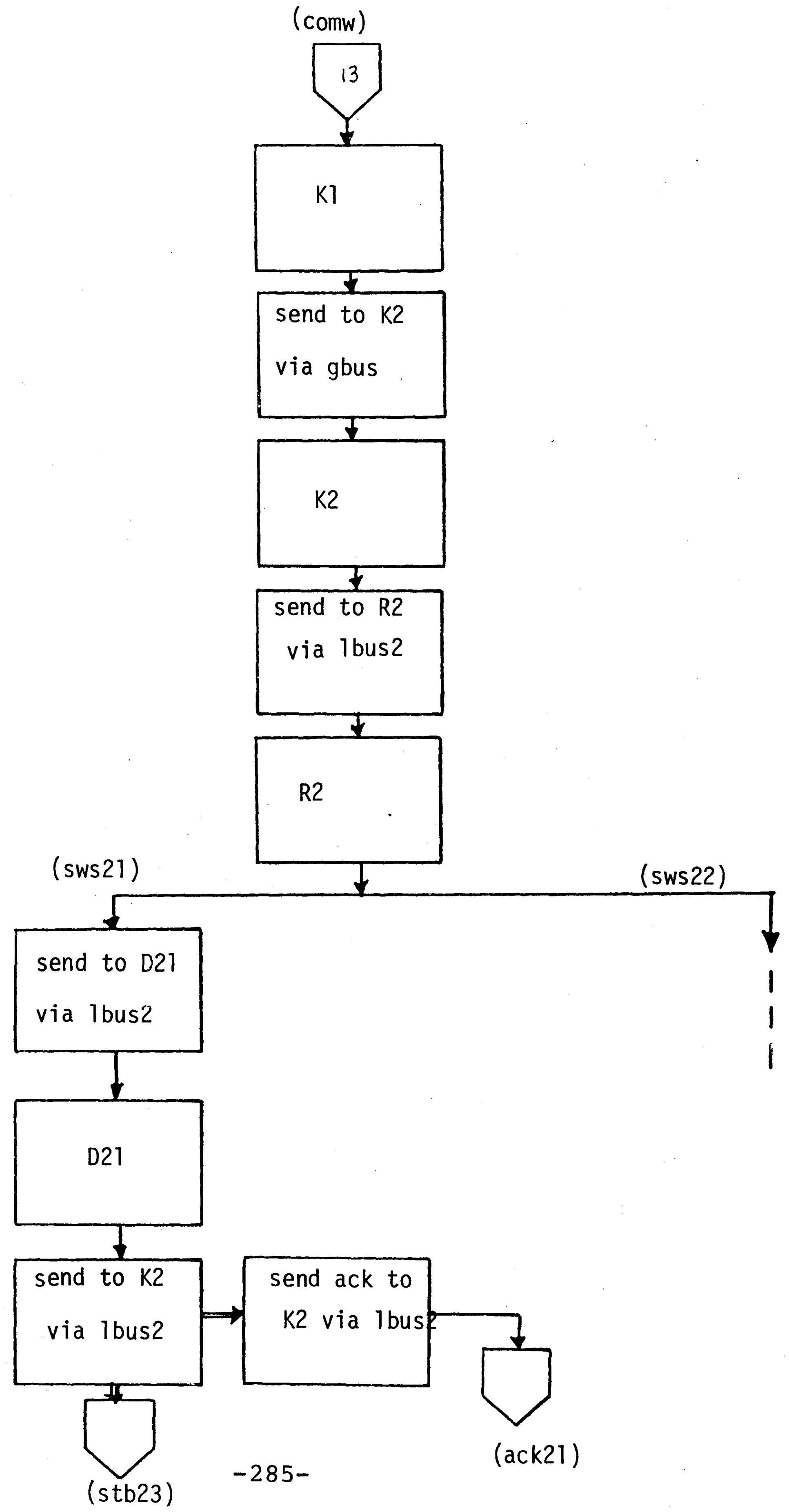

(storl)
8
K1
(wwwl1)
send to
D11 via
lbusl
(wwwl2)
(wwwl3)
(wwwl4)
(wwwl5)
write to
D11
overflow
?
yes
no
send to K1
via lbusl
(starl)
ovl1)
K1
send to K2
via gbus
R2
terminate
K2
send to R2
via lbus2

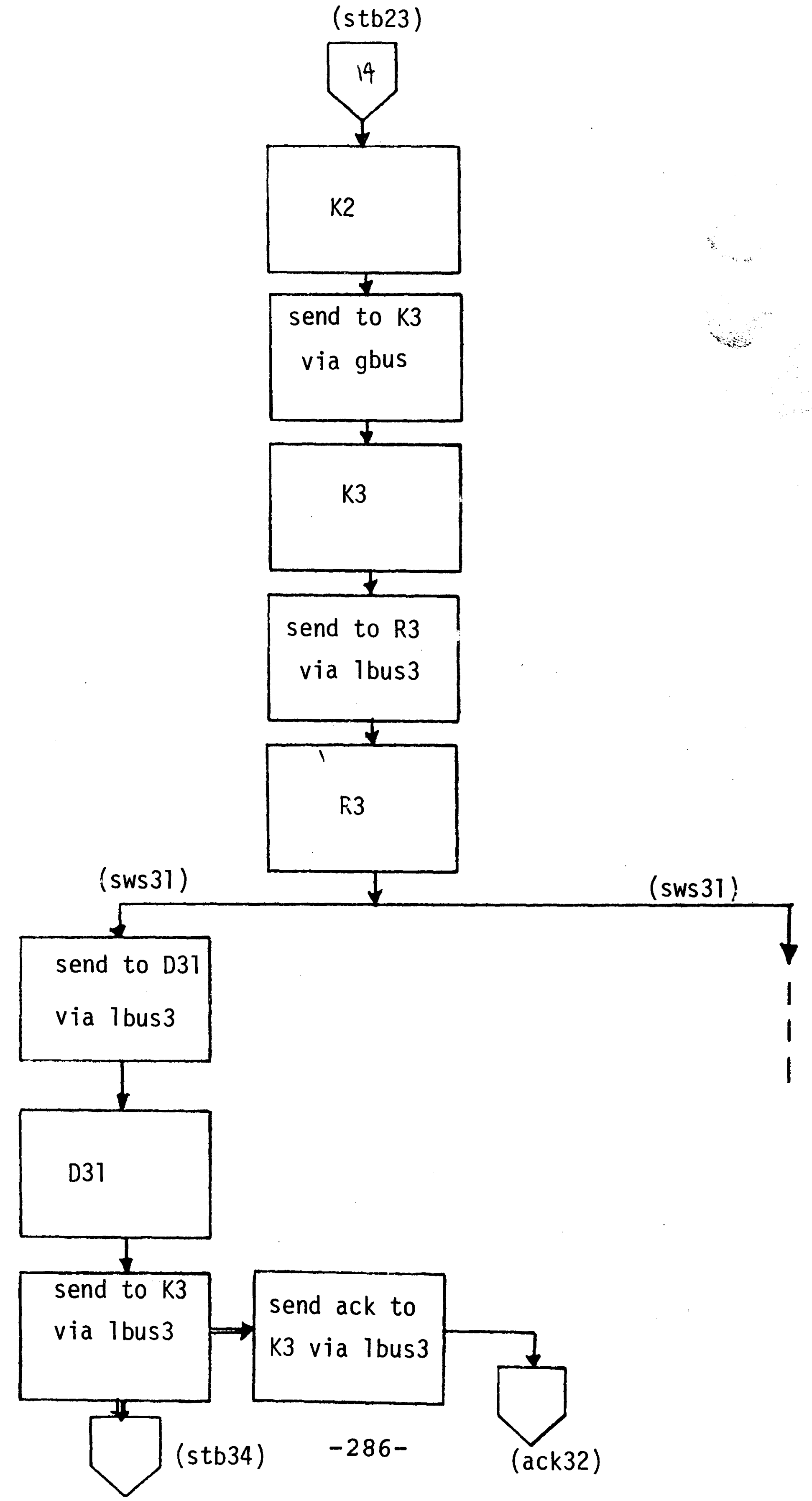

(in13)
9
send to device via lbus3
(rrr31)
(rrr32)
D31
D32
send to K3 via lbus3
send to K3 via lbus3
K3
send to K1 and K2 via gbus
(stor1)
(stor2)

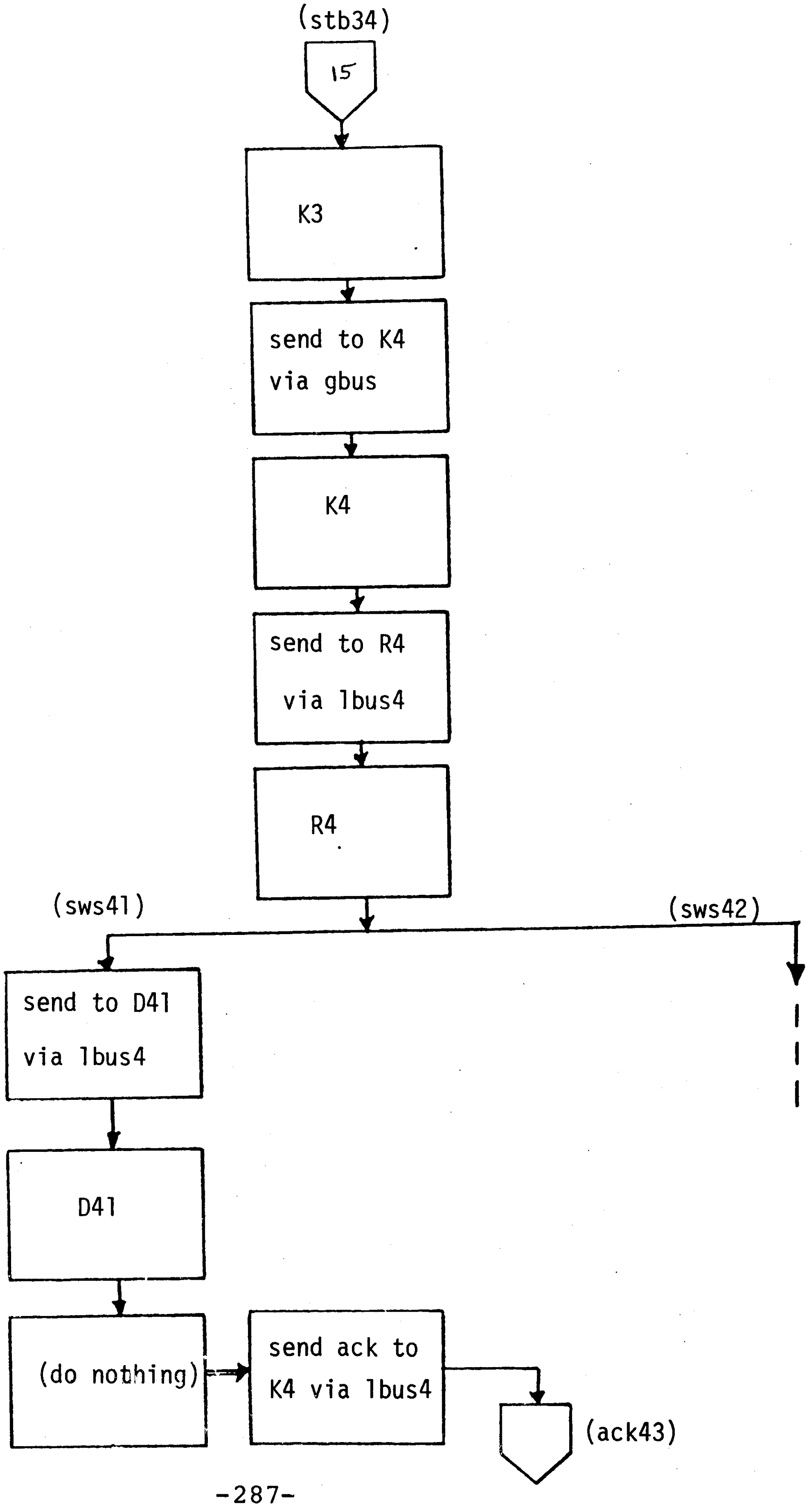

(stor2)
10
K2
send to R2
via lbus2
R2
no
overflow?
yes
terminate
send to K2
via lbus2
K2
send to K3
via gbus
K3
send to R3
via lbus3
R3
terminate
(sss21)
(sss22)
send to D21
via lbus2
store into
D21
terminate

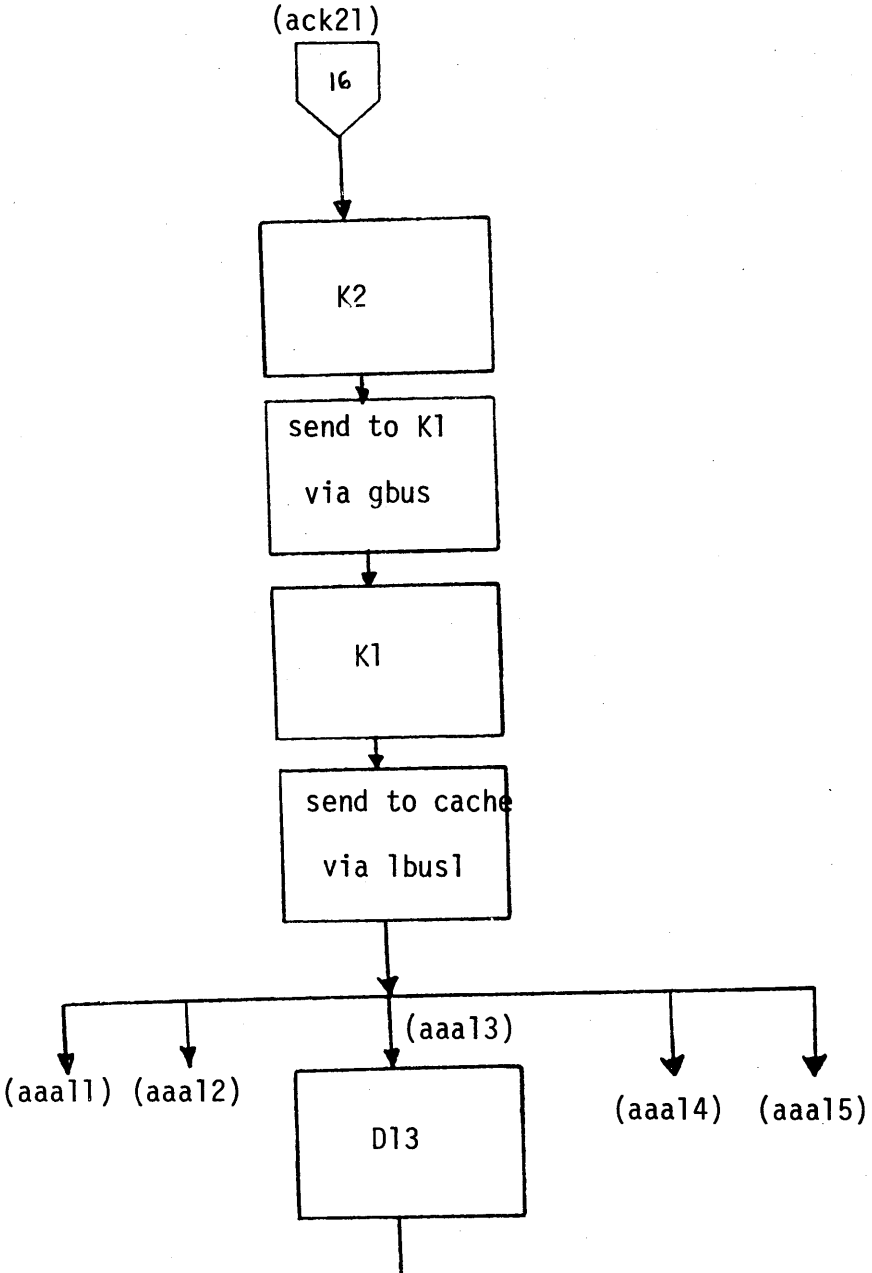

(inl4)
\\
send to device via lbus4
(rrr41)
(rrr42)
D41
D42
send to K4 via lbus4
send to K4 via lbus4
K4
send to K1 K2 and K3 via gbus
(stor1)
(stor2)
(stor3)

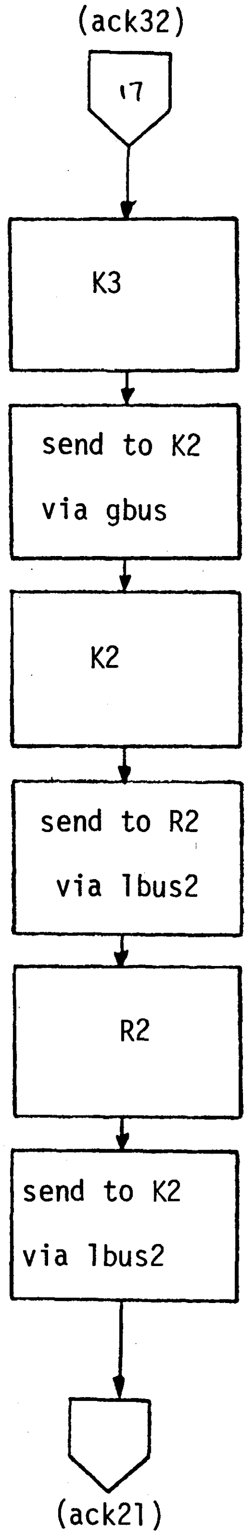

(stor3)
12
K3
send to R2
via lbus3
R3
no
overflow?
yes
terminate
send to K3
via lbus3
K3
send to K4
via gbus
K4
send to R4
via lbus4
R4
terminate
(sss31)
(sss32)
send to D31
via lbus3
D31
terminate

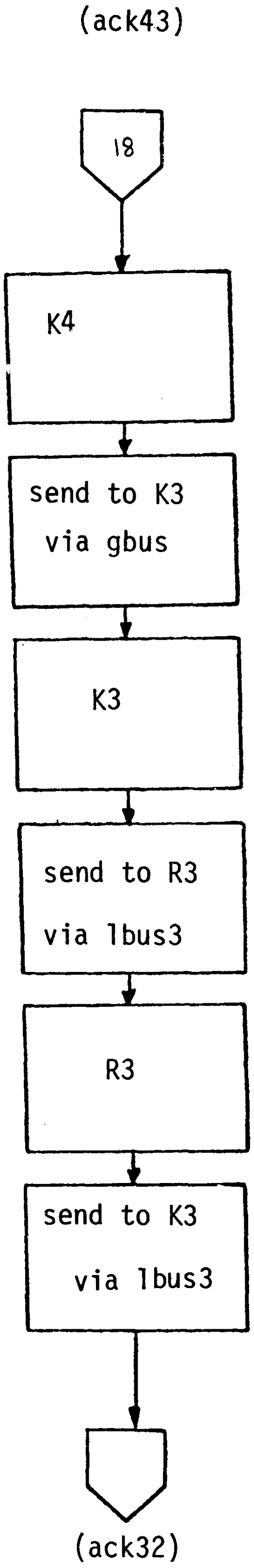

(comw)
13
K1
send to K2
via gbus
K2
send to R2
via lbus2
R2
(sws21)
(sws22)
send to D21
via lbus2
D21
send to K2
via lbus2
send ack to
K2 via lbus2
(stb23)
(ack21)


(stb23)
14
K2
send to K3 via gbus
K3
send to R3 via lbus3
R3
(sws31)
(sws31)
send to D31 via lbus3
D31
send to K3 via lbus3
send ack to K3 via lbus3
(stb34)
(ack32)


(stb34)
15
K3
send to K4 via gbus
K4
send to R4 via lbus4
R4
(sws41)
(sws42)
send to D41 via lbus4
D41
(do nothing)
send ack to K4 via lbus4
(ack43)


(ack21)
16
K2
send to K1
via gbus
K1
send to cache
via lbus1
(aaa11)
(aaa12)
(aaa13)
(aaa14)
(aaa15)
D13
terminate


(ack32)
17
K3
send to K2
via gbus
K2
send to R2
via lbus2
R2
send to K2
via lbus2
(ack21)


(ack43)
18
K4
send to K3
via gbus
K3
send to R3
via lbus3
R3
send to K3
via lbus3
(ack32)

Appendix C

LISTING OF THE P5L4 MODEL

```
//LAM4 JOB LAM,MPROFILE='RETURN',
// PROFILE='LOW',
// TIME=9
//*PASSWORD
//GPSS PROC
//C    EXEC  PGM=DAG01,TIME=&TLIMIT
//STEPLIB DD DSN=POTLUCK.LIBRARY.GPSS.LOAD,DISP=SHR
//DOUTPUT DD SYSOUT=PROFILE=RETURN,DCB=BLKSIZE=931
//DINTERO DD UNIT=SCRATCH,SPACE=(CYL,(1,1)),DCB=BLKSIZE=1880
//DSYMTAB DD UNIT=SCRATCH,SPACE=(CYL,(1,1)),DCB=BLKSIZE=7112
//DREPTGEN DD UNIT=SCRATCH,SPACE=(CYL,(1,1)),DCB=BLKSIZE=800
//DINTWORK DD UNIT=SCRATCH,SPACE=(CYL,(1,1)),DCB=BLKSIZE=2680
// PEND
//STEP1  EXEC  GPSS,PARM=C,TLIMIT=9
//DINPUT1  DD *
       REALLOCATE FUN,5,QUE,10,FAC,50,BVR,200,BLO,2000,VAR,50
       REALLOCATE FSV,50,HSV,10,COM,40000

*************************
*                       *
* TXN PARM USAGE        *
*                       *
*  P1   CPU ID          *
*  P2   TXN ARRIVAL TIME*
*  P3   TXN COMPL TIME  *
*  P4   TXN EXEC TIME   *
*  P11  DUMMY          *
*                       *
*************************

*************************
*                       *
*  MODEL COMPONENTS     *
*                       *
* BUSES: GBUS, LBUS1,.. *
* CACHES: D11,...D15    *
* LEVEL CONTRL: K1,...K4*
* REQ PROCS: R2, .. R4 *
* DEVICES: D21, ...D42  *
* STORAGE : RI, RO      *
* STORAGE : SI, SO      *
* STORAGE : TI, TO      *
* STORAGE : AI, AO      *
* STORAGE : OI, OO      *
*                       *
*************************

*************************
*                       *
* MODEL PARAMETERS      *
*                       *
*************************

       INITIAL    X$MAXMP,10       DEGREE OF MULTIPROG PER CPU
       INITIAL    X$NREAD,500      % READ REQ
       INITIAL    X$NWRIT,500        % WRITE REQ
       INITIAL    X$PIN1,900         CONDITIONAL PROB OF FINDING DATA
       INITIAL    X$PIN2,900         IN A LEVEL GIVEN THAT THE
       INITIAL    X$PIN3,900         DATA IS NOT FOUND IN ANY UPPER
       INITIAL    X$PIN4,1000        LEVEL
       INITIAL    X$POV1,500         PROB OF OVERFLOW
       INITIAL    X$POV2,500
       INITIAL    X$POV3,500
       INITIAL    X$DEX1,10         DEVICE SERVICE TIME
       INITIAL    X$DEX2,100
       INITIAL    X$DEX3,200
       INITIAL    X$DEX4,1000
       INITIAL    X$BEXM,10         BUS SERVICE TIME
       INITIAL    X$BEX1,10
       INITIAL    X$BEX2,80
       INITIAL    X$BEX3,320
       INITIAL    X$REX,20          DIRECTORY LOOK UP
       INITIAL    X$KEX,10          CONTROLLER SERV TIME
       INITIAL    X$RDEX1,30        LOOKUP PLUS READ TIME OF CACHE
       INITIAL    X$TIMER,200000    SIMULATION TIME

*************************
*                       *
* SAVEVALUES            *
*                       *
* NTXN TOTAL TXN PROC.  *
* SUMX TOTAL EXEC TIMES *
* SUMW TOTAL WAIT TIMES *
* SUMT TOTAL ELAPSED TIM*
*                       *
*************************

*************************
*                       *
*  VARIABLES            *
*                       *
*************************

 MRESP FVARIABLE  (X$SUMT/X$NTXN)        MEAN RESP TIME
 TXNT  VARIABLE   P3-P2                  TXN ELAPSED TIME
 TXNW  VARIABLE   P3-P2-P4               TXN WAIT TIME
 TXNX  VARIABLE   P4                     TXN EXEC TIME

*************************
*                       *
*  TABLES              *
*                       *
*************************

 TXNT  TABLE      V$TXNT,100,100,100
 TXNW  TABLE      V$TXNW,100,100,100
 TXNX  TABLE      V$TXNX,100,100,100

*************************
*                       *
*  FUNCTIONS              *
*                         *
**************************

 WICHW FUNCTION   P1,D5
2,WWW11/3,WWW12/4,WWW13/5,WWW14/6,WWW15

 WICHA FUNCTION   P1,D5
2,AAA11/3,AAA12/4,AAA13/5,AAA14/6,AAA15

**************************
*                        *
* STORAGE FOR L(1)       *
* CACHES                 *
*                        *
**************************

       STORAGE     S$RID11,10/S$SID11,2/S$TID11,10/S$AID11,10
       STORAGE     S$RID12,10/S$SID12,2/S$TID12,10/S$AID12,10
       STORAGE     S$RID13,10/S$SID13,2/S$TID13,10/S$AID13,10
       STORAGE     S$RID14,10/S$SID14,2/S$TID14,10/S$AID14,10
       STORAGE     S$RID15,10/S$SID15,2/S$TID15,10/S$AID15,10

**************************
*                        *
* STORAGE FOR DEVICES    *
*                        *
**************************

       STORAGE     S$RID21,10/S$SID21,10/S$TID21,10
       STORAGE     S$RID22,10/S$SID22,10/S$TID22,10
       STORAGE     S$RID31,10/S$SID31,10/S$TID31,10
       STORAGE     S$RID32,10/S$SID32,10/S$TID32,10
       STORAGE     S$RID41,10/S$SID41,10/S$TID41,10
       STORAGE     S$RID42,10/S$SID42,10/S$TID42,10
**************************
*                        *
* STORAGE FOR REQ PROC   *
*                        *
**************************

       STORAGE     S$RIR2,10/S$SIR2,10/S$TIR2,10/S$AIR2,10/S$OIR2,10
       STORAGE     S$RIR3,10/S$SIR3,10/S$TIR3,10/S$AIR3,10/S$OIR3,10
       STORAGE     S$RIR4,10/S$SIR4,10/S$TIR4,10/S$AIR4,10/S$OIR4,10

**************************
*                        *
*  STORAGE FOR K1        *
*                        *
**************************

       STORAGE     S$ROK1,10/S$SOK1,10/S$TIK1,10/S$AIK1,10/S$OOK1,10

**************************
*                        *
* STORAGE FOR K2,K3,K4   *
*                        *
**************************

       STORAGE    S$RIK2,10/S$SIK2,10/S$TIK2,10/S$AIK2,10/S$OIK2,10
       STORAGE    S$RIK3,10/S$SIK3,10/S$TIK3,10/S$AIK3,10/S$OIK3,10
       STORAGE    S$RIK4,10/S$SIK4,10/S$TIK4,10/S$AIK4,10/S$OIK4,10
       STORAGE    S$ROK2,10/S$SOK2,10/S$TOK2,10/S$AOK2,10/S$OOK2,10
       STORAGE    S$ROK3,10/S$SOK3,10/S$TOK3,10/S$AOK3,10/S$OOK3,10
       STORAGE    S$ROK4,10/S$SOK4,10/S$TOK4,10/S$AOK4,10/S$OOK4,10

**************************
*                        *
* BOOLEAN VARIABLES      *
*                        *
**************************

**************************
*                        *
* BV FOR READ-THROUGH    *
*                        *
**************************

 RTOK2 BVARIABLE  FNU$GBUS*SNF$TIK1
 RTOK3 BVARIABLE  FNU$GBUS*SNF$TIK1*SNF$TIK2
 RTOK4 BVARIABLE  FNU$GBUS*SNF$TIK1*SNF$TIK2*SNF$TIK3

**************************
*                        *
* BV FOR L(1)            *
*                        *
**************************

 DKR1   BVARIABLE   FNU$LBUS1*SNF$ROK1
 DKS1   BVARIABLE   FNU$LBUS1*SNF$SOK1
 DKO1   BVARIABLE   FNU$LBUS1*SNF$OOK1
 KDT11  BVARIABLE   FNU$LBUS1*SNF$TID11
 KDT12  BVARIABLE   FNU$LBUS1*SNF$TID12
 KDT13  BVARIABLE   FNU$LBUS1*SNF$TID13
 KDT14  BVARIABLE   FNU$LBUS1*SNF$TID14
 KDT15  BVARIABLE   FNU$LBUS1*SNF$TID15
 KDA11  BVARIABLE   FNU$LBUS1*SNF$AID11
 KDA12  BVARIABLE   FNU$LBUS1*SNF$AID12
 KDA13  BVARIABLE   FNU$LBUS1*SNF$AID13
 KDA14  BVARIABLE   FNU$LBUS1*SNF$AID14
 KDA15  BVARIABLE   FNU$LBUS1*SNF$AID15

**************************
*                        *
* BV FOR INTER LEVEL COM*
*                        *
**************************

 KKR12 BVARIABLE  FNU$GBUS*SNF$RIK2
 KKS12 BVARIABLE  FNU$GBUS*SNF$SIK2
 KKO12 BVARIABLE  FNU$GBUS*SNF$SOK2
 KKT21 BVARIABLE  FNU$GBUS*SNF$TIK1
 KKA21 BVARIABLE  FNU$GBUS*SNF$AIK1
 KKR23 BVARIABLE  FNU$GBUS*SNF$RIK3
 KKS23 BVARIABLE  FNU$GBUS*SNF$SIK3
 KKO23 BVARIABLE  FNU$GBUS*SNF$OIK3
 KKT32 BVARIABLE  FNU$GBUS*SNF$TIK2
 KKA32 BVARIABLE  FNU$GBUS*SNF$AIK2
 KKR34 BVARIABLE  FNU$GBUS*SNF$RIK4
 KKS34 BVARIABLE  FNU$GBUS*SNF$SIK4
 KKO34 BVARIABLE  FNU$GBUS*SNF$OIK4
 KKT43 BVARIABLE  FNU$GBUS*SNF$TIK3
 KKA43 BVARIABLE  FNU$GBUS*SNF$AIK3

**************************
*                        *
* BV FOR L(2) OPS        *
*                        *
**************************

 KRR2   BVARIABLE  FNU$LBUS2*SNF$RIR2
 KRS2   BVARIABLE  FNU$LBUS2*SNF$SIR2
 KRT2   BVARIABLE  FNU$LBUS2*SNF$TIR2
 KRA2   BVARIABLE  FNU$LBUS2*SNF$AIR2
 KRO2   BVARIABLE  FNU$LBUS2*SNF$OIR2
 RDR21  BVARIABLE  FNU$LBUS2*SNF$RID21
 RDS21  BVARIABLE  FNU$LBUS2*SNF$SID21
 RDT21  BVARIABLE  FNU$LBUS2*SNF$TID21
 RDR22  BVARIABLE  FNU$LBUS2*SNF$RID22
 RDS22  BVARIABLE  FNU$LBUS2*SNF$SID22
 RDT22  BVARIABLE  FNU$LBUS2*SNF$TID22
 DKS2   BVARIABLE  FNU$LBUS2*SNF$SOK2
 DKT2   BVARIABLE  FNU$LBUS2*SNF$TOK2
 DKA2   BVARIABLE  FNU$LBUS2*SNF$AOK2
 RKR2   BVARIABLE  FNU$LBUS2*SNF$ROK2
 RKO2   BVARIABLE  FNU$LBUS2*SNF$OOK2
 RKA2   BVARIABLE  FNU$LBUS2*SNF$AOK2

**************************
*                        *
* BV FOR L(3) OPS        *
*                        *
**************************

 KRR3   BVARIABLE  FNU$LBUS3*SNF$RIR3
 KRS3   BVARIABLE  FNU$LBUS3*SNF$SIR3
 KRT3   BVARIABLE  FNU$LBUS3*SNF$TIR3
 KRA3   BVARIABLE  FNU$LBUS3*SNF$AIR3
 KRO3   BVARIABLE  FNU$LBUS3*SNF$OIR3
 RDR31  BVARIABLE  FNU$LBUS3*SNF$RID31
 RDS31  BVARIABLE  FNU$LBUS3*SNF$SID31
 RDT31  BVARIABLE  FNU$LBUS3*SNF$TID31
 RDR32  BVARIABLE  FNU$LBUS3*SNF$RID32
 RDS32  BVARIABLE  FNU$LBUS3*SNF$SID32
 RDT32  BVARIABLE  FNU$LBUS3*SNF$TID32
 DKS3  BVARIABLE  FNU$LBUS3*SNF$SOK3
 DKT3  BVARIABLE  FNU$LBUS3*SNF$TOK3
 DKA3  BVARIABLE  FNU$LBUS3*SNF$AOK3
 RKR3  BVARIABLE  FNU$LBUS3*SNF$ROK3
 RKA3  BVARIABLE  FNU$LBUS3*SNF$AOK3
 RKO3  BVARIABLE  FNU$LBUS3*SNF$OOK3

*************************
*                       *
* BV FOR L(4) OPS       *
*                       *
*************************

 KRR4  BVARIABLE  FNU$LBUS4*SNF$RIR4
 KRS4  BVARIABLE  FNU$LBUS4*SNF$SIR4
 KRO4  BVARIABLE  FNU$LBUS4*SNF$OIR4
 RDR41 BVARIABLE  FNU$LBUS4*SNF$RID41
 RDS41 BVARIABLE  FNU$LBUS4*SNF$SID41
 RDR42 BVARIABLE  FNU$LBUS4*SNF$RID42
 RDS42 BVARIABLE  FNU$LBUS4*SNF$SID42
 DKT4  BVARIABLE  FNU$LBUS4*SNF$TOK4
 DKA4  BVARIABLE  FNU$LBUS4*SNF$AOK4

*************************
*                       *
*    MACROS             *
*                       *
*************************

*************************
*                       *
* MACRO -USE            *
* #A  FACILITY         *
* #B  USAGE TIME       *
*                       *
*************************

 USE   STARTMACRO
       SEIZE      #A
       ADVANCE    #B
       ASSIGN     4+,#B
       RELEASE    #A
       ENDMACRO

*************************
*                       *
*  MACRO - SEND         *
*                       *
* #A  FROM             *
* #B  TO               *
* #C  VIA              *
* #D  TRANSIT TIME     *
* #E  BV FOR SEND OP  *
*                       *
*************************
```

```
 SEND  STARTMACRO
       TEST E      #E,1
       ENTER       #B
       SEIZE       #C
       ADVANCE     #D
       ASSIGN      4+,#D
       RELEASE     #C
       LEAVE       #A
       ENDMACRO

*************************
*                       *
*  MACRO - FINI         *
*                       *
*************************

 FINI  STARTMACRO
       MARK        3
       SAVEVALUE   NTXN+,1
       SAVEVALUE   SUMX+,V$TXNX
       SAVEVALUE   SUMW+,V$TXNW
       SAVEVALUE   SUMT+,V$TXNT
       SAVEVALUE   MRESP,V$MRESP
       ASSIGN      1,0
       ASSIGN      2,0
       ASSIGN      3,0
       ASSIGN      4,0
       ENDMACRO

**------------------------------------------------------------*
*                                                             *
*          BEGIN SIMULATION                                   *
*                                                             *
**------------------------------------------------------------*

       SIMULATE
*************************
*                       *
*   CPU #1              *
*                       *
*************************

       RMULT       3,5,7,9,11,13,15,17

 CPU1  GENERATE    ,,,X$MAXMP,,,F
 STAR1 PRIORITY    9                 SET HIGH P FOR NEW TXN
       MARK        2                 ARRIVAL TIME
       ASSIGN      1,1               CPU ID
       TRANSFER    .X$NREAD,WWW1,RRR1
 RRR1  TRANSFER    .X$PIN1,NIN11,RIN11

*************************
*                       *
* DATA IS IN DATA CACHE *
*                        *
**************************

 RIN11 ENTER      RID11           PUT TXN IN READ REQ BUFFER
 USE   MACRO      DRP11,X$RDEX1   SEARCH AND READ CACHE
       LEAVE      RID11           FREE BUFFER
 FINI  MACRO
       TRANSFER   ,STAR1          A NEW TXN

**************************
*                        *
* DATA IS NOT IN CACHE   *
*                        *
**************************

 NIN11 ENTER      RID11           PUT IN READ REQ BUFFER
 USE   MACRO      DRP11,X$REX     SEARCH DIRECTORY
       PRIORITY   0               RESET PRIORITY
 SEND  MACRO      RID11,ROK1,LBUS1,X$BEXM,BV$DKR1

       TRANSFER   ,COMR           TO COMMON CODE FOR READ

**************************
*                        *
* WRITE REQUEST TO CACHE*
*                        *
**************************

 WWW1  ENTER      SID11           PUT TXN IN WRITE REQ BUFFER

 USE   MACRO      DRP11,X$RDEX1    WRITE DATA IN CACHE

       PRIORITY   0               RESET TXN PRIORITY
 SEND  MACRO      SID11,SOK1,LBUS1,X$BEX1,BV$DKS1

       SPLIT      1,COMW

 FINI  MACRO

       TRANSFER   ,STAR1          A NEW TXN

**************************
*                        *
*   CPU #2              *
*                        *
**************************

 CPU2  GENERATE   ,,,X$MAXMP,,,F
 STAR2 PRIORITY   9               SET HIGH P FOR NEW TXN
       MARK       2               ARRIVAL TIME
       ASSIGN     1,2             CPU ID
       TRANSFER   .X$NREAD,WWW2,RRR2
 RRR2  TRANSFER   .X$PIN1,NIN12,RIN12

**************************
```

```
*                        *
* DATA IS IN DATA CACHE *
*                        *
*************************

 RIN12 ENTER     RID12            PUT TXN IN READ REQ BUFFER
 USE   MACRO     DRP12,X$RDEX1    SEARCH AND READ CACHE
       LEAVE     RID12            FREE BUFFER
 FINI  MACRO
       TRANSFER  ,STAR2           A NEW TXN

*************************
*                        *
* DATA IS NOT IN CACHE  *
*                        *
*************************

 NIN12 ENTER     RID12            PUT IN READ REQ BUFFER
 USE   MACRO     DRP12,X$REX      SEARCH DIRECTORY
       PRIORITY  0                RESET PRIORITY
 SEND  MACRO     RID12,ROK1,LBUS1,X$BEXM,BV$DKR1

       TRANSFER  ,COMR            TO COMMON CODE FOR READ

*************************
*                        *
* WRITE REQUEST TO CACHE*
*                        *
*************************

 WWW2  ENTER     SID12            PUT TXN IN WRITE REQ BUFFER

 USE   MACRO     DRP12,X$RDEX1     WRITE DATA IN CACHE

       PRIORITY  0                RESET TXN PRIORITY
 SEND  MACRO     SID12,SOK1,LBUS1,X$BEX1,BV$DKS1

       SPLIT     1,COMW

 FINI  MACRO

       TRANSFER  ,STAR2           A NEW TXN

*************************
*                        *
*   CPU #3              *
*                        *
*************************

 CPU3  GENERATE  ,,,X$MAXMP,,,F
 STAR3 PRIORITY  9                SET HIGH P FOR NEW TXN
       MARK      2                ARRIVAL TIME
       ASSIGN    1,3              CPU ID
       TRANSFER  .X$NREAD,WWW3,RRR3
 RRR3  TRANSFER  .X$PIN1,NIN13,RIN13
```

```
*************************
*                       *
* DATA IS IN DATA CACHE *
*                       *
*************************

 RIN13 ENTER      RID13             PUT TXN IN READ REQ BUFFER
 USE   MACRO      DRP13,X$RDEX1     SEARCH AND READ CACHE
       LEAVE      RID13             FREE BUFFER
 FINI  MACRO
       TRANSFER   ,STAR3            A NEW TXN

*************************
*                       *
* DATA IS NOT IN CACHE  *
*                       *
*************************

 NIN13 ENTER      RID13             PUT IN READ REQ BUFFER
 USE   MACRO      DRP13,X$REX       SEARCH DIRECTORY
       PRIORITY   C                 RESET PRIORITY
 SEND  MACRO      RID13,ROK1,LBUS1,X$BEXM,BV$DKR1

       TRANSFER   ,COMR             TO COMMON CODE FOR READ

*************************
*                       *
* WRITE REQUEST TO CACHE*
*                       *
*************************

 WWW3  ENTER      SID13             PUT TXN IN WRITE REQ BUFFER

 USE   MACRO      DRP13,X$RDEX1      WRITE DATA IN CACHE

       PRIORITY   0                 RESET TXN PRIORITY

 SEND  MACRO      SID13,SOK1,LBUS1,X$BEX1,BV$DKS1

       SPLIT      1,COMW

 FINI  MACRO

       TRANSFER   ,STAR3

*************************
*                       *
*   CPU #4             *
*                       *
*************************

 CPU4  GENERATE   ,,,X$MAXMP,,,F
 STAR4 PRIORITY   9                 SET HIGH P FOR NEW TXN
       MARK       2                 ARRIVAL TIME
       ASSIGN     1,4                 CPU ID
       TRANSFER   .X$NREAD,WWW4,RRR4
 RRR4  TRANSFER   .X$PIN1,NIN14,RIN14

*************************
*                       *
* DATA IS IN DATA CACHE *
*                       *
*************************

 RIN14 ENTER      RID14               PUT TXN IN READ REQ BUFFER
 USE   MACRO      DRP14,X$RDEX1       SEARCH AND READ CACHE
       LEAVE      RID14               FREE BUFFER
 FINI  MACRO
       TRANSFER   ,STAR4              A NEW TXN

*************************
*                       *
* DATA IS NOT IN CACHE  *
*                       *
***** ********************

 NIN14 ENTER      RID14               PUT IN READ REQ BUFFER
 USE   MACRO      DRP14,X$REX         SEARCH DIRECTORY
       PRIORITY   0                   RESET PRIORITY
 SEND  MACRO      RID14,ROK1,LBUS1,X$BEXM,BV$DKR1

       TRANSFER   ,COMR               TO COMMON CODE FOR READ

*************************
*                       *
* WRITE REQUEST TO CACHE*
*                       *
*************************

 WWW4  ENTER      SID14               PUT TXN IN WRITE REQ BUFFER

 USE   MACRO      DRP14,X$RDEX1        WRITE DATA IN CACHE

       PRIORITY   0                   RESET TXN PRIORITY

 SEND  MACRO      SID14,SOK1,LBUS1,X$BEX1,BV$DKS1

       SPLIT      1,COMW

 FINI  MACRO

       TRANSFER   ,STAR4              A NEW TXN

*************************
*                       *
*   CPU #5             *
*                       *
*************************
```

```
 CPU5  GENERATE  ,,,X$MAXMP,,,F
 STAR5 PRIORITY  9                SET HIGH P FOR NEW TXN
       MARK      2                ARRIVAL TIME
       ASSIGN    1,5              CPU ID
       TRANSFER  .X$NREAD,WWW5,RRR5
 RRR5  TRANSFER  .X$PIN1,NIN15,RIN15

*************************
*                       *
* DATA IS IN DATA CACHE *
*                       *
*************************

 RIN15 ENTER     RID15            PUT TXN IN READ REQ BUFFER
 USE   MACRO     DRP15,X$RDEX1    SEARCH AND READ CACHE
       LEAVE     RID15            FREE BUFFER
 FINI  MACRO
       TRANSFER  ,STAR5           A NEW TXN

*************************
*                       *
* DATA IS NOT IN CACHE  *
*                       *
*************************

 NIN15 ENTER     RID15            PUT IN READ REQ BUFFER
 USE   MACRO     DRP15,X$REX      SEARCH DIRECTORY
       PRIORITY  0                RESET PRIORITY
 SEND  MACRO     RID15,ROK1,LBUS1,X$BEXM,BV$DKR1

       TRANSFER  ,COMR            TO COMMON CODE FOR READ

*************************
*                       *
* WRITE REQUEST TO CACHE*
*                       *
*************************

 WWW5  ENTER     SID15            PUT TXN IN WRITE REQ BUFFER

 USE   MACRO     DRP15,X$RDEX1     WRITE DATA IN CACHE

       PRIORITY  0                RESET TXN PRIORITY

 SEND  MACRO     SID15,SOK1,LBUS1,X$BEX1,BV$DKS1

       SPLIT     1,COMW

 FINI  MACRO

       TRANSFER  ,STAR5           A NEW TXN

*---------------------------------------------------------------*
*                                                               *
*   COMMON CODE FOR READ REQUEST                                *
*                                                                    *
*--------------------------------------------------------------------*

 COMR  ASSIGN      11,0

 USE   MACRO       KRP1,X$KEX

 SEND  MACRO       ROK1,RIK2,GBUS,X$BEXM,BV$KKR12

 USE   MACRO       KRP2,X$KEX

 SEND  MACRO       RIK2,RIR2,LBUS2,X$BEXM,BV$KRR2

 USE   MACRO       RRP2,X$REX

       TRANSFER    .X$PIN2,NIN2,RIN2
 NIN2  ASSIGN      11,0

 SEND  MACRO       RIR2,ROK2,LBUS2,X$BEXM,BV$RKR2

 USE   MACRO       KRP2,X$KEX

 SEND  MACRO       ROK2,RIK3,GBUS,X$BEXM,BV$KKR23

 USE   MACRO       KRP3,X$KEX

 SEND  MACRO       RIK3,RIR3,LBUS3,X$BEXM,BV$KRR3

 USE   MACRO       RRP3,X$REX

       TRANSFER    .X$PIN3,NIN3,RIN3
 NIN3  ASSIGN      11,0

 SEND  MACRO       RIR3,ROK3,LBUS3,X$BEXM,BV$RKR3

 USE   MACRO       KRP3,X$KEX

 SEND  MACRO       ROK3,RIK4,GBUS,X$BEXM,BV$KKR34

 USE   MACRO       KRP4,X$KEX

 SEND  MACRO       RIK4,RIR4,LBUS4,X$BEXM,BV$KRR4

 USE   MACRO       RRP4,X$REX

       TRANSFER    ,RIN4

*--------------------------------------------------------------------*
*                                                                    *
*  READ DATA IS FOUND IN L(2)                                        *
*                                                                    *
*--------------------------------------------------------------------*

 RIN2  TRANSFER    .5,RRR21,RRR22
```

```
*************************
*                       *
* DATA IS IN D21        *
*                       *
*************************

 BRR21 ASSIGN    11,0

 SEND  MACRO     RIR2,RID21,LBUS2,X$BEXM,BV$RDR21

 USE   MACRO     DRP21,X$DEX2

 SEND  MACRO     RID21,TOK2,LBUS2,X$BEX1,BV$DKT2

       TRANSFER  ,RTF2

*************************
*                       *
* DATA IS IN D22        *
*                       *
*************************

 BRR22 ASSIGN    11,0

 SEND  MACRO     RIR2,RID22,LBUS2,X$BEXM,BV$RDR22

 USE   MACRO     DRP22,X$DEX2

 SEND  MACRO     RID22,TOK2,LBUS2,X$BEX1,BV$DKT2

       TRANSFER  ,RTF2

*************************
*                       *
* READ-THROUGH TO L(1)  *
*                       *
*************************

 RTF2  ASSIGN    11,0

 USE   MACRO     KRP2,X$KEX

 SEND  MACRO     TOK2,TIK1,GBUS,X$BEX1,BV$RTOK2

*-----------------------------------------------------------*
*                                                           *
* STORE DATA INTO L(1) AS RESULT OF A READ-THROUGH          *
*                                                           *
*-----------------------------------------------------------*

 STOR1 ASSIGN    11,0

 USE   MACRO     KRP1,X$KEX

       SPLIT     1,FN$WICHW,1
       TERMINATE

*************************
*                       *
* RT STORE INTO D11     *
*                       *
*************************

 WWW11 ASSIGN      11,0

 SEND  MACRO       TIK1,TID11,LBUS1,X$BEX1,BV$KDT11

 USE   MACRO       DRP11,X$DEX1

       TRANSFER    .X$POV1,NOV11,OVL11
 NOV11 LEAVE       TID11

 FINI  MACRO

       TRANSFER    ,STAR1

 OVL11 SPLIT       1,OVF11

 FINI  MACRO

       TRANSFER    ,STAR1
 OVF11 ASSIGN      11,0

 SEND  MACRO       TID11,OOK1,LBUS1,X$BEXM,BV$DKO1

       TRANSFER    ,OVL1

*************************
*                       *
* RT STORE INTO D12     *
*                       *
*************************

 WWW12 ASSIGN      11,0

 SEND  MACRO       TIK1,TID12,LBUS1,X$BEX1,BV$KDT12

 USE   MACRO       DRP12,X$DEX1

       TRANSFER    .X$POV1,NOV12,OVL12
 NOV12 LEAVE       TID12

 FINI  MACRO

       TRANSFER    ,STAR2

 OVL12 SPLIT       1,OVF12

 FINI  MACRO
```

```
        TRANSFER    ,STAR2
 OVF12 ASSIGN      11,0

 SEND  MACRO       TID12,OOK1,LBUS1,X$BEXM,BV$DKO1

       TRANSFER    ,OVL1

*************************
*                       *
* RT STORE INTO D13     *
*                       *
*************************

 WWW13 ASSIGN      11,0

 SEND  MACRO       TIK1,TID13,LBUS1,X$BEX1,BV$KDT13

 USE   MACRO       DRP13,X$DEX1

       TRANSFER    .X$POV1,NOV13,OVL13
 NOV13 LEAVE       TID13

 FINI  MACRO

       TRANSFER    ,STAR3

 OVL13 SPLIT       1,OVF13

 FINI  MACRO

       TRANSFER    ,STAR3
 OVF13 ASSIGN      11,0

 SEND  MACRO       TID13,OOK1,LBUS1,X$BEXM,BV$DKO1

       TRANSFER    ,OVL1

*************************
*                       *
* RT STORE INTO D14     *
*                       *
*************************

 WWW14 ASSIGN      11,0

 SEND  MACRO       TIK1,TID14,LBUS1,X$BEX1,BV$KDT14

 USE   MACRO       DRP14,X$DEX1

       TRANSFER    .X$POV1,NOV14,OVL14
 NOV14 LEAVE       TID14

 FINI  MACRO

       TRANSFER    ,STAR4
 OVL14 SPLIT      1,OVF14

 FINI  MACRO

       TRANSFER   ,STAR4
 OVF14 ASSIGN     11,0

 SEND  MACRO      TID14,OOK1,LBUS1,X$BEXM,BV$DKO1

       TRANSFER   ,OVL1

*************************
*                       *
* RT STORE INTO D15     *
*                       *
*************************

 WWW15 ASSIGN     11,0

 SEND  MACRO      TIK1,TID15,LBUS1,X$BEX1,BV$KDT15

 USE   MACRO      DRP15,X$DEX1

       TRANSFER   .X$POV1,NOV15,OVL15
 NOV15 LEAVE      TID15

 FINI  MACRO

       TRANSFER   ,STAR5

 OVL15 SPLIT      1,OVF15

 FINI  MACRO

       TRANSFER   ,STAR5
 OVF15 ASSIGN     11,0

 SEND  MACRO      TID15,OOK1,LBUS1,X$BEXM,BV$DKO1

       TRANSFER   ,OVL1

*************************
*                       *
* HANDLE OVF FROM L(1)  *
*                       *
*************************

 OVL1  ASSIGN     11,0

 USE   MACRO      KRP1,X$KEX

 SEND  MACRO      OOK1,OIK2,GBUS,X$BEXM,BV$KKO12

 USE   MACRO      KRP2,X$KEX
 SEND  MACRO       OIK2,OIR2,LBUS2,X$BEXM,BV$KRO2

 USE   MACRO       RRP2,X$REX

       LEAVE       OIR2
       TERMINATE

*----------------------------------------------------------------*
*                                                                *
*  READ DATA IS FOUND IN L(3)                                    *
*                                                                *
*----------------------------------------------------------------*

 RIN3  TRANSFER    .5,RRR31,RRR32

*************************
*                       *
* DATA IS IN D31        *
*                       *
*************************

 RRR31 ASSIGN      11,0

 SEND  MACRO       RIR3,RID31,LBUS3,X$BEXM,BV$RDR31

 USE   MACRO       DRP31,X$DEX3

 SEND  MACRO       RID31,TOK3,LBUS3,X$BEX2,BV$DKT3

       TRANSFER    ,RTF3

*************************
*                       *
* DATA IS IN D32        *
*                       *
*************************

 RRR32 ASSIGN      11,0

 SEND  MACRO       RIR3,RID32,LBUS3,X$BEXM,BV$RDR32

 USE   MACRO       DRP32,X$DEX3

 SEND  MACRO       RID32,TOK3,LBUS3,X$BEX2,BV$DKT3

       TRANSFER    ,RTF3

*************************
*                       *
* RT TO L(1) AND L(2)   *
*                       *
*************************

 RTF3  ASSIGN      11,0
```

```
USE   MACRO      KRP3,X$KEX

      TEST E     BV$RTOK3,1
      ENTER      TIK1
      ENTER      TIK2
      SEIZE      GBUS
      ADVANCE    X$BEX2
      ASSIGN     4+,X$BEX2
      RELEASE    GBUS
      LEAVE      TOK3
      SPLIT      1,STOR1
      SPLIT      1,STOR2
      TERMINATE

*--------------------------------------------------------------*
*                                                              *
* STORE DATA INTO L(2) AS RESULT OF A READ-THROUGH             *
*                                                              *
*--------------------------------------------------------------*

STOR2 ASSIGN     11,0

USE   MACRO      KRP2,X$KEX

SEND  MACRO      TIK2,TIR2,LBUS2,X$BEX2,BV$KRT2

USE   MACRO      RRP2,X$REX

      SPLIT      1,OVH2
      TRANSFER   .5,SSS21,SSS22

*************************
*                       *
* STORE INTO D21        *
*                       *
*************************

SSS21 ASSIGN     11,0

SEND  MACRO      TIR2,TID21,LBUS2,X$BEX2,BV$RDT21

USE   MACRO      DRP21,X$DEX2

      LEAVE      TID21
      TERMINATE

*************************
*                       *
* STORE INTO D22        *
*                       *
*************************

SSS22 ASSIGN     11,0
```

```
 SEND  MACRO      TIR2,TID22,LBUS2,X$BEX2,BV$RDT22

 USE   MACRO      DRP22,X$DEX2

       LEAVE      TID22
       TERMINATE

*************************
*                       *
* OVERFLOW HANDLING     *
*                       *
*************************

 OVH2  TRANSFER   .X$POV2,NOVL2,OVL2
 OVL2  TEST E     BV$RKO2,1
       ENTER      OOK2
       SEIZE      LBUS2
       ADVANCE    X$BEXM
       ASSIGN     4+,X$BEXM
       RELEASE    LBUS2

 SEND  MACRO      OOK2,OIK3,GBUS,X$BEXM,BV$KKO23

 USE   MACRO      KRP3,X$KEX

 SEND  MACRO      OIK3,OIR3,LBUS3,X$BEXM,BV$KRO3

 USE   MACRO      RRP3,X$REX

       LEAVE      OIR3
 NOVL2 TERMINATE

*--------------------------------------------------------------*
*                                                              *
* READ DATA IS FOUND IN L(4)                                   *
*                                                              *
*--------------------------------------------------------------*

 RIN4  TRANSFER   .5,RRR41,RRR42

*************************
*                       *
* DATA IS IN D41        *
*                       *
*************************

 RRR41 ASSIGN     11,0

 SEND  MACRO      RIR4,RID41,LBUS4,X$BEXM,BV$RDR41

 USE   MACRO      DRP41,X$DEX4

 SEND  MACRO      RID41,TOK4,LBUS4,X$BEX3,BV$DKT4

       TRANSFER   ,RTF4
```

```
*************************
*                       *
* DATA IS IN D42        *
*                       *
*************************

 BRR42 ASSIGN      11,0

 SEND  MACRO       RIR4,RID42,LBUS4,X$BEXM,BV$RDR42

 USE   MACRO       DRP42,X$DEX4

 SEND  MACRO       RID42,TOK4,LBUS4,X$BEX3,BV$DKT4

       TRANSFER    ,RTF4

*************************
*                       *
* RT TO L(1),L(2),L(3)  *
*                       *
*************************

 RTF4  ASSIGN      11,0

 USE   MACRO       KRP4,X$KEX

       TEST E      BV$RTOK4,1
       ENTER       TIK1
       ENTER       TIK2
       ENTER       TIK3
       SEIZE       GBUS
       ADVANCE     X$BEX3
       ASSIGN      4+,X$BEX3
       RELEASE     GBUS
       LEAVE       TOK4
       SPLIT       1,STOR1
       SPLIT       1,STOR2
       SPLIT       1,STOR3
       TERMINATE

*------------------------------------------------------------*
*                                                            *
*  STORE INTO L(3) AS A RESULT OF READ-THROUGH               *
*                                                            *
*------------------------------------------------------------*

 STOR3 ASSIGN      11,0

 USE   MACRO       KRP3,X$KEX

 SEND  MACRO       TIK3,TIR3,LBUS3,X$BEX3,BV$KRT3

 USE   MACRO       RRP3,X$REX
```

```
       SPLIT      1,OVH3
       TRANSFER   .5,SSS31,SSS32

*************************
*                       *
* STORE INTO D31        *
*                       *
*************************

SSS31  ASSIGN     11,0

 SEND  MACRO      TIR3,TID31,LBUS3,X$BEX3,BV$RDT31

 USE   MACRO      DRP31,X$DEX3

       LEAVE      TID31
       TERMINATE

*************************
*                       *
* STORE INTO D32        *
*                       *
*************************

 SSS32 ASSIGN     11,0

 SEND  MACRO      TIR3,TID32,LBUS3,X$BEX3,BV$RDT32

 USE   MACRO      DRP32,X$DEX3

       LEAVE      TID32
       TERMINATE

*************************
*                       *
* OVERFLOW HANDLING     *
*                       *
*************************

 OVH3  TRANSFER   .X$POV3,NOVL3,OVL3
 OVL3  TEST E     BV$RKO3,1
       ENTER      OOK3
       SEIZE      LBUS3
       ADVANCE    X$BEXM
       ASSIGN     4+,X$BEXM
       RELEASE    LBUS3

 SEND  MACRO      OOK3,OIK4,GBUS,X$BEXM,BV$KKO34

 USE   MACRO      KRP4,X$KEX

 SEND  MACRO      OIK4,OIR4,LBUS4,X$BEXM,BV$KRO4

 USE   MACRO      RRP4,X$REX
       LEAVE      OIR4
 NOVL3 TERMINATE

*----------------------------------------------------------------*
*                                                                *
*  COMMON CODE FOR STORE-BEHIND                                  *
*                                                                *
*----------------------------------------------------------------*

 COMW  ASSIGN      11,0

 USE   MACRO       KRP1,X$KEX

 SEND  MACRO       SOK1,SIK2,GBUS,X$BEX1,BV$KKS12

 USE   MACRO       KRP2,X$KEX

 SEND  MACRO       SIK2,SIR2,LBUS2,X$BEX1,BV$KRS2

 USE   MACRO       RRP2,X$REX

       TRANSFER    .5,SWS21,SWS22

*************************
*                       *
* SB WRITE INTO D21     *
*                       *
*************************

 SWS21 ASSIGN      11,0

 SEND  MACRO       SIR2,SID21,LBUS2,X$BEX1,BV$RDS21

 USE   MACRO       DRP21,X$DEX2

 SEND  MACRO       SID21,SOK2,LBUS2,X$BEX2,BV$DKS2

       SPLIT       1,STB23
       ENTER       AOK2
       TRANSFER    ,ACK21

*************************
*                       *
* SB WRITE INTO D22     *
*                       *
*************************

 SWS22 ASSIGN      11,0

 SEND  MACRO       SIR2,SID22,LBUS2,X$BEX1,BV$RDS22

 USE   MACRO       DRP22,X$DEX2

 SEND  MACRO       SID22,SOK2,LBUS2,X$BEX2,BV$DKS2

       SPLIT       1,STB23
       ENTER      AOK2
       TRANSFER   ,ACK21

*-------------------------------------------------------------*
*                                                             *
* STORE-BEHIND TO L(3)                                        *
*                                                             *
*-------------------------------------------------------------*

 STB23 ASSIGN     11,0

 USE   MACRO      KRP2,X$KEX

 SEND  MACRO      SOK2,SIK3,GBUS,X$BEX2,BV$KKS23

 USE   MACRO      KRP3,X$KEX

 SEND  MACRO      SIK3,SIR3,LBUS3,X$BEX2,BV$KRS3

 USE   MACRO      RRP3,X$REX

       TRANSFER   .5,SWS31,SWS32

*************************
*                       *
* SB WRITE INTO D31     *
*                       *
*************************

 SWS31 ASSIGN     11,0

 SEND  MACRO      SIR3,SID31,LBUS3,X$BEX2,BV$RDS31

 USE   MACRO      DRP31,X$BEX3

 SEND  MACRO      SID31,SOK3,LBUS3,X$BEX3,BV$DKS3

       SPLIT      1,STB34
       ENTER      AOK3
       TRANSFER   ,ACK32

*************************
*                       *
* SB WRITE INTO D32     *
*                       *
*************************

 SWS32 ASSIGN     11,0

 SEND  MACRO      SIR3,SID32,LBUS3,X$BEX2,BV$RDS32

 USE   MACRO      DRP32,X$DEX3

 SEND  MACRO      SID32,SOK3,LBUS3,X$BEX3,BV$DKS3
```

```
      SPLIT       1,STB34
      ENTER       AOK3
      TRANSFER    ,ACK32

*------------------------------------------------------------*
*                                                            *
*  STORE-BEHIND TO L(4)                                      *
*                                                            *
*------------------------------------------------------------*

 STB34 ASSIGN     11,0

 USE   MACRO      KRP3,X$KEX

 SEND  MACRO      SOK3,SIK4,GBUS,X$BEX3,BV$KKS34

 USE   MACRO      KRP4,X$KEX

 SEND  MACRO      SIK4,SIR4,LBUS4,X$BEX3,BV$KRS4

 USE   MACRO      RRP4,X$REX

       TRANSFER   .5,SWS41,SWS42

*************************
*                       *
* SB WRITE INTO D41     *
*                       *
*************************

 SWS41 ASSIGN     11,0

 SEND  MACRO      SIR4,SID41,LBUS4,X$BEX3,BV$RDS41

 USE   MACRO      DRP41,X$DEX4

 SEND  MACRO      SID41,AOK4,LBUS4,X$BEXM,BV$DKA4

       TRANSFER   ,ACK43

*************************
*                       *
* SB WRITE INTO D42     *
*                       *
*************************

 SWS42 ASSIGN     11,0

 SEND  MACRO      SIR4,SID42,LBUS4,X$BEX3,BV$RDS42

 USE   MACRO      DRP42,X$DEX4

 SEND  MACRO      SID42,AOK4,LBUS4,X$BEXM,BV$DKA4

       TRANSFER   ,ACK43
```

```
*--------------------------------------------------------------*
*                                                              *
*  ACK FROM L(4) TO L(3)                                       *
*                                                              *
*--------------------------------------------------------------*

 ACK43 ASSIGN      11,0

 USE   MACRO       KRP4,X$KEX

 SEND  MACRO       AOK4,AIK3,GBUS,X$BEXM,BV$KKA43

 USE   MACRO       KRP3,X$KEX

 SEND  MACRO       AIK3,AIR3,LBUS3,X$BEXM,BV$KRA3

 USE   MACRO       RRP3,X$REX

************************
*                      *
* FORWARD THE ACK UP   *
*                      *
************************

 SEND  MACRO       AIR3,AOK3,LBUS3,X$BEXM,BV$RKA3

 USE   MACRO       KRP3,X$KEX

 SEND  MACRO       AOK3,AIK2,GBUS,X$BEXM,BV$KKA32

 USE   MACRO       KRP2,X$KEX

 SEND  MACRO       AIK2,AIR2,LBUS2,X$BEXM,BV$KRA2

 USE   MACRO       RRP2,X$REX

       LEAVE       AIR2
       TERMINATE

*--------------------------------------------------------------*
*                                                              *
* ACK FROM L(3) TO L(2)                                        *
*                                                              *
*--------------------------------------------------------------*

 ACK32 ASSIGN      11,0

 USE   MACRO       KRP3,X$KEX

 SEND  MACRO       AOK3,AIK2,GBUS,X$BEXM,BV$KKA32

 USE   MACRO       KRP2,X$KEX

 SEND  MACRO       AIK2,AIR2,LBUS2,X$BEXM,BV$KRA2
 USE   MACRO      RRP2,X$REX

 SEND  MACRO      AIR2,AOK2,LBUS2,X$BEXM,BV$RKA2

       TRANSFER   ,ACK21

*------------------------------------------------------------*
*                                                            *
*  ACK FROM L(2) TO L(1)                                     *
*                                                            *
*------------------------------------------------------------*

 ACK21 ASSIGN     11,0

 USE   MACRO      KRP2,X$KEX

 SEND  MACRO      AOK2,AIK1,GBUS,X$BEXM,BV$KKA21

 USE   MACRO      KRP1,X$KEX

       SPLIT      1,FN$WICHA,1
       TERMINATE

**************************
*                        *
* ACK HANDLED BY D11     *
*                        *
**************************

 AAA11 ASSIGN     11,0

 SEND  MACRO      AIK1,AID11,LBUS1,X$BEXM,BV$KDA11

 USE   MACRO      DRP11,X$REX

       LEAVE      AID11
       TERMINATE

**************************
*                        *
* ACK HANDLED BY D12     *
*                        *
**************************

 AAA12 ASSIGN     11,0

 SEND  MACRO      AIK1,AID12,LBUS1,X$BEXM,BV$KDA12

 USE   MACRO      DRP12,X$REX

       LEAVE      AID12
       TERMINATE

**************************
```

```
*                        *
* ACK HANDLED BY D13     *
*                        *
**************************

 AAA13 ASSIGN     11,0

 SEND  MACRO      AIK1,AID13,LBUS1,X$BEXM,BV$KDA13

 USE   MACRO      DRP13,X$REX

       LEAVE      AID13
       TERMINATE

**************************
*                        *
* ACK HANDLED BY D14     *
*                        *
**************************

 AAA14 ASSIGN     11,0

 SEND  MACRO      AIK1,AID14,LBUS1,X$BEXM,BV$KDA14

 USE   MACRO      DRP14,X$REX

       LEAVE      AID14
       TERMINATE

**************************
*                        *
* ACK HANDLED BY D15     *
*                        *
**************************

 AAA15 ASSIGN     11,0

 SEND  MACRO      AIK1,AID15,LBUS1,X$BEXM,BV$KDA15

 USE   MACRO      DRP15,X$REX

       LEAVE      AID15
       TERMINATE

*----------------------------------------------------------------*
*                                                                *
*  SIMULATION CONTROL                                            *
*                                                                *
*----------------------------------------------------------------*

       GENERATE   X$TIMER
       TERMINATE  1
       START      1
       END
```

BIOGRAPHIC NOTE

Chat-Yu Lam (林澤宇) was born in Swatow, Kwuntung Province, China, on December 18, 1951. He emigrated to HongKong in 1959 where he completed grade school at the Chi Tak Public School in 1964 and completed high school at the Queen Elizabeth School in 1970.

He attended Massachusetts Institute of Technology in September 1970 and graduated in June 1974 with a B.Sc. in Electrical Engineering. During this time, he was a systems programmer for the M.I.T. Departmental Information System and was in the Cambridge Project JANUS Database Management System design team.

In September 1974 he attended Northwestern University and graduated with a M.Sc. in Computer Science in June 1976. During this time he was also a software system analyst at A.B. Dick Company, Chicago.

He began his doctoral program at the Sloan School of Management, M.I.T., in September 1976, majoring in Management Information Systems. While at the Sloan School, he was a research assistant for the NEEMIS Project, the RADC Decision Support Systems Project, and the INFOPLEX Project. He was a teaching assistant for the Database Systems, Operating Systems, and Systems Programming courses at the Sloan School. He was also in the MIMS Database Management System development team at the MITROL Inc., Waltham, Massachusetts.

PUBLICATIONS

1. 'Properties of Storage Hierarchy Systems with Multiple Page Sizes and Redundant Data', *ACM Trans. on Database Systems*, *4* *3* (September 1979) (with S. Madnick)

2. 'INFOPLEX Data Base Computer Architecture - Concepts and Directions', M.I.T. Sloan School Working Paper No. 1046-79 (C.I.S.R. Working Paper No. 41), 1979 (with S. Madnick)

3. 'Composite Information Systems - A New Concept in Information Systems', M.I.T. Sloan School Working Paper No. 993-78 (C.I.S.R. Working paper No. 35), 1978 (with S. Madnick)

4. 'The GMIS 2 Architecture', M.I.T. Energy Laboratory Working Paper No. MIT-EL-77-014WP, 1977 (with J. Lamb et. al.)